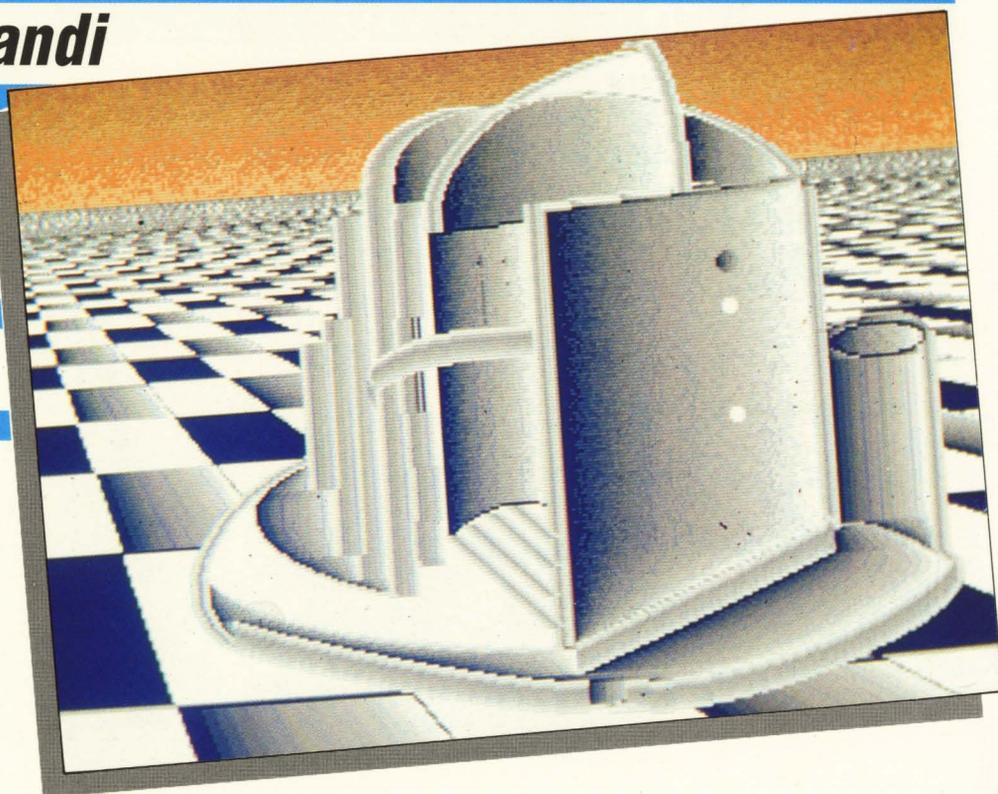
L. 7.000 - S.FR. 10,50

NUMERO 5 1989

LA RIVISTA PER I PROGRAMMATORI DI AMIGA

- Impariamo a programmare in Assembler
- Breakpoint
- Programmare in ARexx
- I dischi di Fred Fish
- Come usare la BridgeBoard
- Cambiamenti nel Disk Filing System
- Devpac Hisoft
- Lattice C 5.0
- I Device Handler
- Introduzione a Robbs
- Estendiamo l'uso dei nostri comandi
- Un'occhiata al software per l'A500

TAXE PERCUE (TASSA RISCOSSA) MILANO FERR. CORR. SPEDIZIONE IN ABB. POST. GRUPPO IV/70

Vuoi sapere proprio tutto sui migliori videogiochi?

LA GRANDE GUIDA A TUTTI I GIOCHI ELETTRONICI E NON

La prima vera grande guida indipendente a tutti i migliori giochi per computer, console, giochi da bar e altro ancora.

In ogni numero trovi:

- più di 30 giochi al microscopio
- novità e anteprime
- i game da bar più gettonati
- recensioni dei giochi più famosi
- Nintendomania.

Scegli il meglio: scegli Jackson.

Ladri! 2

La ripresa del titolo dello scorso numero non è dovuta al fatto che sia diventato di successo, come è stato per Rocky, ma è semplicemente dovuta alla necessità di approfondire temi, diciamo, che sono cari alla testata. Fin dall'inizio abbiamo cercato di segnalare quei mali congeniti che affliggono il mondo dell'informatica consumer italiana. Non che l'informatica "personale", indicando con questo il mondo che gravita attorno ai personal computer usati in ufficio, non soffra degli stessi problemi, solo che c'è una maggior quantità di denaro e una ignoranza più diffusa.
Dopo aver lanciato accuse a tutti, dai commercianti, ai produttori e ai centri di assistenza, nonché agli utenti che si tirano la zappa sui piedi, abbiamo ricevuto poche risposte: un centro di assistenza, alcune lettere e, indirettamente, da una ditta produttrice di software.
Il centro di assistenza autorizzato da Commodore si chiama Computer Lab e si trova a Milano. Si è lamentato del fatto che non tutti i centri assistenza siano gestiti da persone improvvisate e con mezzi tecnici e scadenti. Siamo andati a trovarli e l'impressione che ne abbiamo ricavato è stata buona: sia il cliente che il computer vengono trattati con cura e professionalità. Sicuramente è un centro di assistenza che si distacca dalla media.
La ditta si chiama Cloanto, è di Udine e ha prodotto un word processor di nome C1-TEXT. Il software è di ottimo livello e confrontabile con prodotti d'oltre oceano: non ne esiste comunque uno che gli sia superiore. È pensato per il mercato europeo e la versione italiana sarà imbattibile per diverso tempo. Il prezzo è di 69.000 lire, compresa la spedizione espressa a casa se non lo si riesce a trovare in un negozio, con un esauriente manuale in dotazione. Non sarebbe un furto, copiarlo?
Le lettere parlano invece genericamente di varie disavventure e chiedono quali possano essere i rimedi a una situazione del genere. Come credo sia stato chiaro dall'editoriale del numero precedente, ritengo che la situazione da terzo mondo in cui si trova l'informatica italiana sia dovuta al radicato "malcostume" di copiare il software e, addirittura di rivenderlo, causando un doppio danno al produttore! Per rendesi conto di quanto sia estesa questa pratica è sufficiente riferirsi a una vicenda della scorsa primavera quando venne accertato che diverse grosse aziende italiane, taciamo il nome per stendere un velo pietoso, utilizzano in massima parte software copiato.
Gli italiani, si sa, sono il popolo che "fatta la legge, trovano l'inganno"; figuriamoci se la legge non esiste nemmeno. Proprio la legge di tutela del software è il punto centrale del problema: senza quella non è possibile rimediare alla situazione se non con azioni sporadiche di qualche entusiasta (leggi Cloanto, Computer Lab e chi altro). Siamo rimasti l'unico paese industrializzato senza uno straccio di legge e peggio di noi si sta solo in America latina. Nel panorama dell'editoria tecnica italiana è un fiorire di editorialisti che invocano leggi adeguate per il proprio settore. L'esempio più lampante è quello di una rivista di informatica ben nota dove il proprio direttore da anni (due ?) invoca, coprendo di ridicolo a più riprese la SIP e i ministeri competenti, una legge e un po' di sensibilità riguardo l'utilizzo dei modem e della telematica personale. Spero, sebbene sia convinto del contrario, di non dover scrivere, così come l'illustre collega, ancora due anni di editoriali prima che venga approvata una legge minima.
A chi poi si fosse scandalizzato per la parola "mafia" presente nell'editoriale del numero scorso, e ne avrebbe ben donde, mi permetta di ricordagli che Transactor è una rivista un po' "scomoda" per il movimento che ruota attorno ad Amiga e che siamo costretti a muoverci tra gelosie, incomprensioni e bastoni tra le ruote. Se poi si fosse ulteriormente dubbiosi basta contare le pagine di pubblicità presenti in questo numero o in quelli scorsi.

Marco Ottolini

DIRETTORE RESPONSABILE
Paolo Reina

DIRETTORE TECNICO
Marco Ottolini

TRADUZIONI E REDAZIONE
Leonardo Fei, Stefano Simonelli,
Paolo Toccaceli, Giovanna Traversa

GRAFICA
Roberto Pessina

IMPAGINAZIONE ELETTRONICA
Roberto Gibertini

STAMPA
Grafica F.B.M. Gorgonzola (MI)

AUTORIZZAZIONE ALLA PUBBLICAZIONE
Trib. di Milano n. 866 del 20/12/88

AREA CONSUMER
PUBLISHER
Filippo Canavese

DIREZIONE
Via Pola, 9 - 20124 Milano
Tel. 02/69401
Telefax 02/6948238

REDAZIONE, PUBLICITA', AMMINISTRAZIONE, ABBONAMENTI
Via Rossellini, 12 - 20124 Milano
Tel. 02/69401
Telex: 333436 GEJIT
Telefax: 02/6948489

CONCESSIONARIO ESCLUSIVO
SODIP - Via Zuretti, 25 - 20125 Milano

MAGAZZINO
Via Gasparotto, 15 - 20092 Cinisello B. (MI)
Tel. 02/61222527-6187736

SEDE LEGALE
Via Pietro Mascagni, 14 - 20122 Milano

PREZZO DELLA RIVISTA: L. 7.000
NUMERI ARRETRATI: L. 14.000
ABBONAMENTO ANNUO (6 numeri): L. 34.000
ESTERO: L. 68.000

Spedizione in Abb. Post. Gruppo IV/70

PUBLISHER
David H. Beatty

EDITOR
Mike Todd

CONTRIBUTING WRITERS
S. Ahlstrom, S. Ballantine, C.B. Blish, J. Butterfield, Betty Clay,
D. Curtis, M. Dillon, A. Finkel, C. Innes, C. Gray, P. Kivolowitz,
R. Mariani, B. Nesbitt, R. Peck, L. Phillips, B. Rakosky,
J. Toebes, V.A. Wagner, D. Wood

Sommario

Associato al

Testata aderente al C.S.S.T. non soggetta a certificazione obbligatoria in quanto la presenza pubblicitaria è inferiore al 10%

Lettere e Bit

Gentile Redazione, sono un vostro appassionato lettore possessore di un Amiga 1000. Tralasciando i meritatissimi complimenti per non farla troppo lunga, passerei subito alla questione: leggendo la vostra rubrica "LETTERE..." del n. 3 di Transactor ho notato un problema esposto dal Sig. Massimo Rainato nella sua lettera. Il problema riguarda la mancanza di certa documentazione sulle LIBRARY di Amiga; mi spiego meglio: il Sig. Rainato (come penso molti altri possessori /programmatori di Amiga) non ha trovato alcun problema nel reperire la documentazione riguardante le famose librerie in questione, il loro significato tecnico e la loro utilità, i problemi sorgono però quando le si devono utilizzare, come fare?
Sfruttando la mia poca esperienza e alcune giornate di pioggia sono riuscito a scoprire (ma forse non è questa la parola giusta...) le famose routine disponibili in ogni libreria. La cosa, che può sembrare abbastanza ardua, si rivela di una semplicità che fa paura. Le librerie di cui parlo sono quelle disponibili da BASIC, visto che i miei "esperimenti" sono stati tutti eseguiti in questo linguaggio, ma in seguito darò alcune buone notizie anche per quello che riguarda il C.
Passiamo dunque ai fatti, tutti noi abbiamo nel nostro dischetto EXTRAS il famoso AmigaBASIC, in questo dischetto è presente anche una directory molto importante: FDx.x (la x.x sta a indicare la versione del nostro EXTRAS). Se visualizziamo il contenuto di questa directory ci accorgiamo che sono presenti dei file che terminano tutti in "_lib.fd", sono proprio queste le famose librerie che possono essere utilizzate dal BASIC, o meglio, sono queste trasformate in file ".bmap" (per fare ciò, tutti sapranno che bisogna utilizzare l'utility presente nel dischetto Extras, nella directory BasicDemos di nome "ConvertFD") che dovranno essere presenti nello stesso dischetto in cui risiede l'AmigaBASIC.
Dopo questa parentesi non del tutto necessaria vorrei tornare ai file ".fd" di prima e alla loro importanza: è proprio in questi che sono presenti le famose routine disponibili in BASIC; se noi facciamo un veloce dump di queste (basta il comando Type) ci accorgeremo che appariranno come un intero file ASCII in cui sono contenute ordinatamente le routine disponibili ed i parametri necessari.
Sembra impossibile, vero ?!?
Ma diamo un'occhiata più da vicino a questi file: le prime due righe sono riservate al sistema: la prima dà il nome della libreria, la seconda non ho ancora capito a cosa serva. Dopo ciò seguirà una favolosa lista di routine disponibili (che saranno la gioia di tutti i possessori di Amiga) seguita ognuna dai parametri necessari a dai registri che utilizzerà (opzionali quando si utilizza la routine); alla fine del file ci sarà quasi sempre la parola "##end". C'è da notare che alcune routine non saranno disponibili: sono quelle precedute da "##private".
Qui sopra ho descritto come conoscere le potenze utilizzabili, ma non posso prolungarmi nel descrivere come utilizzarle perché la carta è tiranna, quindi mi limiterò solo ad accennare alle librerie con il C. Queste sono molto più complesse che nel BASIC (cosa abbastanza immaginabile) e cercherò di sintetizzare al massimo: ogni compilatore C (io uso il Lattice) è dotato di una directory "include", qui sono presenti le famose ed estese librerie disponibili; facendo il dump di ognuna di queste si possono scoprire le varie Structure utilizzabili in un programma C; certo, prima occorre aprire la libreria con il comando "Include <nomelibreria>, e per quanto riguarda il resto vi rimando alla pratica.
Io non me ne intendo molto di C, ma ho provato con successo quello che ho appena asserito.
Per concludere vorrei porre i miei cordiali saluti al Sig.Rainato a cui consiglio di fare delle stampe dei file .fd per averli sempre a disposizione, e vorrei porre i miei saluti alla Redazione di Transactor, una grandissima rivista che spero in seguito migliori la sua puntualità.

Massimo Marcelli, Corciano

Lo spirito d'avventura è sicuramente da lodare quando l'obiettivo delle ricerche è avere una maggior padronanza di Amiga. Purtroppo non è stato sufficiente ne a te, caro Massimo, ne ad altri che hanno solo fatto un po' di confusione pur essendo vicini alla soluzione dell'arcano. In realtà i file che hai esaminato non sono le librerie, che tanti sonni tolgono ai programmatori, ma il mezzo per arrivare a usare le librerie.
Le librerie sono delle collezioni di routine che si trovano direttamente in ROM (nel Kickstart), oppure sul dischetto del Workbench nella directory libs (LIBS: sotto AmigaDOS). I linguaggi di programmazione, siano essi il C, l'Assembler o il BASIC, non "sanno" come usare le librerie, finché non gli si forniscono le informazioni necessarie. I file .fd, per esempio, contengono i nomi delle routine contenute nelle diverse librerie e l'elenco dei registri nei quali devono essere posti i parametri. L'utility ConvertFD provvede poi, per mezzo di queste informazioni, a generare i file .bmap corretti, usati infine dall'AmigaBASIC per richiamare le routine delle librarie.
Il C, per farla semplice, funziona in maniera simile, solo che tali informazioni si trovano nei file .lib contenuti, generalmente, in una directory "lib" (il nome avrebbe dovuto suggerire qualcosa al nostro "esploratore"). Oltre a permettere l'accesso alle routine di sistema i file .lib contengono, in realtà è questa la loro funzione, delle routine fornite a corredo del compilatore quali printf, scanf, ecc.
I file .h, detti include in gergo, del C contengono invece una serie di istruzioni C valide che, per mezzo della direttiva #include <nome file> vengono inglobate all'interno di un programma C. Al loro interno si trovano solitamente dichiarazioni di costanti (#define) e definizioni di tipi di variabili che possono essere utilizzate in diversi programmi.
Per quanto riguarda l'Assembler riferitivi agli articoli di Jim Butterfield sull'argomento che, a partire da questo numero, saranno pubblicati su Transactor.

SoftMail

Vendita per corrispondenza di programmi originali ed accessori per computers.

VIA NAPOLEONA 16 - 22100 COMO - TEL. (031) 30.01.74

® SoftMail è un marchio registrato da Lago snc

Presentiamo in questa pagina alcune tra le ultime novità disponibili dal catalogo SoftMail. Ecco alcune informazioni utili per utilizzare il servizio SoftMail: è possibile effettuare ordini telefonicamente SOLO se è già stata effettuata una spedizione a proprio nome ed è stata regolarmente ritirata. Dal secondo in poi accettiamo ordini telefonici. Chi desidera notizie sulla disponibilità ed i prezzi dei prodotti che non compaiono in questa lista può chiamare lo (031)30.01.74 dalle 14:30 alle 18:00. SoftMail può organizzare la consegna anche tramite corriere: interpellateci per maggiori informazioni. Oltre alle ultime novità qui esposte, SoftMail offre l'intero catalogo delle seguenti case: Aegis, Cinemaware, EA, Microprose, Rainbird, SSI, SSG, Sublogic.

Ti è arrivato il catalogo a *colori* SoftMail? Se vuoi riceverlo *gratuitamente*, telefonaci subito!

031/30.01.74

ACCESSORI

Copritastiera A500	25.000
Final cartdrige III	110.000
Flicker master	29.000
Joy. Slik stick	16.500
Joy. SpeedKing	29.000
Joy. SpeedKing Autof.	33.000
Joy. Tac 2	29.000
Joy. Tac 5	39.000
MouseMat tappetino	22.500
Coprimouse	20.000
Portamouse	12.500
Portadischi 3" (30)	34.000
Portadischi 5" (40)	37.000
SlimLine (tastiera 64)	49.000
Voice messenger C64	60.000

LIBRI/HINTS & TIPS

Alternate city clue specificare 8/16 bit	18.000
Bard's tale I clue	22.500
Bard's tale II clue	25.000
Bard's tale III clue	25.000
Black cauldron clue	18.000
Deathlord clue	telef.
Dungeon master clue	25.000
Elite clue	18.000
Graphic adv.creator hint	7.500
Mars saga clue	telef.
Might & Magic clue	25.000
Pool of radiance cle	20.000
Quest for clues	39.000
Sentinel world clue	25.000
Starflight clue	25.000
Ultima V clue	22.500
Wasteland clue	16.500

AMIGA

Heroes lance hints disk	*telef.*
Bard's tale hints disk	*telef.*
Aegis draw 2000	telef.
Aegis images	69.000
Aegis impact	125.000
Aegis sonix	110.000
Aegis videotitiler 1.1	185.000
Alternate reality: the city	69.000
Animation multiplane	125.000
Armageddon man	49.000
Audiomaster	85.000
Audiomaster II	telef.
Baal	29.000
Barbarian II	telef.
Batman	29.000
Battlechess	49.000
Black cauldron	9.000
Blitzkrieg at Ardennes	59.000
California games	25.000
Carrier command	49.000
Cell animator	telef.
Chrono quest	65.000
Comic setter	139.000
Comic setter art disks	telef.
Corruption	49.000
Cosmic pirates	telef.
Crazy cars	29.000
Def con 5	59.000
Defender of the crown	59.000
Deluxe print II	99.000
DigiPaint	99.000
DigiView GOLD	telef.
Director	99.000
Disk drive esterno	299.000
Dos2Dos	75.000
Dragon's lair (1 MByte, 6 disks)	75.000
Driller TUTTO italiano	59.000
Dungeon master (1 MB)	59.000
Elite	45.000
Empire	49.000
F16 Falcon	59.000
Fantavision	125.000
Fire & forget	39.000
Fish	45.000
Flight simulator II	99.000
Scenery disks	telef.
Galileo 2.0	99.000
Garfield	49.000
Gauntlet II	25.000
Grabbit	49.000
Hellfire attack	39.000
International soccer	39.000
Italy 90 soccer	39.000
Incr.s.sphere	49.000
Jet & Japan bundle	110.000
Joan of Arc	25.000
King of Chicago	49.000
LED storm	25.000
Legend of the sword	45.000
Life cycles vol.1	39.000
Light!Camera!Action!	99.000
Lords of the Rising Sun	telef.
Major motion	39.000
Maxiplan plus	299.000
Menace	39.000
Mickey Mouse	25.000
Modeler 3D	129.000
Murder on Atlantic	59.000
Nigel Mansell grand prix	49.000
Obliterator	45.000
Offshore warrior	39.000
Outrun	25.000
P.O.W.	59.000
PacMania	25.000
Phantom fighter	45.000
Pioneer plague	59.000
President is missing	49.000
ProSound designer	79.000
ProSound w/hardware	199.000
Prowrite 2.0	179.000
Publisher plus	125.000
Reach for the stars	59.000
Rebel...Chickamauga	telef.
Return to Genesis	39.000
Rocket Ranger	59.000
Roger Rabbit	69.000
SEUCK	telef.
Sex vixens in space	55.000
Sculpt/Animate 4D	telef.
Sentinel	49.000
Sinbad	59.000
Skyfox II	59.000
Starfleet	59.000
Starglider II	49.000
Superman	39.000
Sword of sodan	69.000
The bard's tale II	49.000
The works!	249.000
Three Stooges	59.000
Tracker	49.000
TV Sports: Football	59.000
Ultima IV	49.000
Universal military sim.	45.000
Data disks 1 e 2	telef.
Victory road	39.000
Videoscape 3D 2.0	250.000
Designs disks	telef.
VIP professional	199.000
Virus	29.000
Virus infection protection	89.000
WB Extras	69.000
Whirligig	39.000
Willow	59.000
World Cl. Leaderboard	25.000
Zak McKracken	59.000
Zing keys	25.000
Zoetrope	199.000
Zoom	45.000
3 Demon	145.000

COMMODORE 64 CASS.

After burner	18.000
Barbarian II	18.000
Batman	18.000
Dragon ninja	15.000
Exploding fist+	15.000
G.I.hero	15.000
Italy 90 soccer	18.000
Last Ninja II	25.000
MicroSoccer	telef.
Robocop	18.000
R-type	18.000
Serve & volley	22.000
Shoot'em up constr. kit	25.000
Stunt bike simulator	5.000
Tiger road	15.000
Total eclipse (ital.)	15.000
4 soccer simulator	18.000

COMMODORE 64 DISCO

ADD: Heroes of the lance	telef.
ADD: Pool of radiance	59.000
American civil war III	49.000
Barbarian II	25.000
Deathlord	35.000
Driller	29.000
Echelon (con LipStick)	59.000
Eternal dagger	35.000
Fast break	29.000
Fish	29.000
Game over I+II	25.000
GeoPublisher	100.000
Grand Prix Circuit	telef.
Gulf strike	49.000
Home video producer	69.000
Italy 90 soccer	25.000
LED storm	15.000
Mars saga	35.000
McArhtu's war	49.000
MicroSoccer	telef.
Neuromancer	39.000
One on one II	29.000
Powerplay hockey	29.000
Red storm rising	49.000
Rocket Ranger	telef.
Shoot'em up constr. kit	35.000
Stealth mission	75.000
T.K.O.	29.000
The bard's tale III	35.000
Total eclipse (ital.)	20.000
Ultima V	49.000
Wasteland	39.000
Wec Les Mans	21.000
Who framed Roger Rabbit	telef.
Zak Mc Kracken	39.000
4 soccer simulator	25.000

COMMODORE 128 DISCO (128K, 80 COL.)

"C" Language	99.000
Basic 8.0	75.000
GEOS 128	telef.
Mach 128	125.000

OFFERTE SPECIALI

(FINO AD ESAURIMENTO)

		Valore	A sole
FIREBIRD SPECIAL			
AMIGA	Bubble Bobble+Whirligig+ Enlightenment	107.000	75.000
C64 cass.	Savage+Intensity+Sentinel	48.000	33.000
C64 disco	Savage+Intensity+Sentinel	65.000	39.000
AEGIS ENTERTAINMENT			
AMIGA	Arazok's Tomb+Ports of Call	120.000	89.000

DEFENDER OF THE CROWN C64: cass. 12.000 disco 15.000

... SOFTMAIL TI DA' DI PIU'.

BUONO D'ORDINE **da inviare a: LAGO DIVISIONE SOFTMAIL, VIA NAPOLEONA 16, 22100 COMO, FAX (031) 30.02.14, TEL (031) 30.01.74**

Desidero ricevere i seguenti articoli:

☐ **Addebitate l'importo sulla mia CARTASI/MASTERCARD/VISA/AMERICAN EXPRESS nr.** ____________ **scadenza** ____________

☐ **Pagherò al postino in contrassegno**

Titolo del programma	Computer	Cassetta/disco/accessorio	Prezzo
		Contributo spese di spedizione Lit.	5.000
		TOTALE LIT.	

ORDINE MINIMO LIT. 20.000 (SPESE ESCLUSE)

Cognome e nome ____________

Indirizzo ____________ **Nr.** ______

CAP ______ **Città** ____________ **Prov.** ______ **Telefono** ____________

FIRMA (SE MINORENNE QUELLA DI UN GENITORE)

VERRANNO EVASI SOLO GLI ORDINI FIRMATI ____________

TA00 5

Guida per gli utenti della BridgeBoard

(o di un Amiga 1000 col SideCar)

di Dan Schein

Dan Schein ha fatto parte del Commodore Amiga Technical Support e ora lavora nel dipartimento di informatica di un ospedale della Pennsylvania. I suoi articoli tecnici relativi ad Amiga sono già apparsi su diverse pubblicazioni.

Se gli si aggiunge una BridgeBoard, Amiga 2000 diventa un computer con due diverse *personalità*. Una tale combinazione rende la macchina differente da tutte quelle apparse in precedenza? Per la verità, direi proprio di sì. Prima di addentrarci nella sostanza di questo articolo vorrei fare una precisazione: un Amiga 2000 con BridgeBoard è equivalente da un punto di vista funzionale a un Amiga 1000 con SideCar, quindi nel seguito quando mi riferirò a un Amiga 2000 con BridgeBoard, le affermazioni varranno nella stessa identica maniera anche per un 1000 col SideCar. Se, poi, possedete proprio un 1000 col SideCar e vi accorgete che vi manca qualcuna delle utility e dei programmi di cui parlerò in questo articolo, vi suggerirei di andare dal vostro negoziante per acquistare una copia dell'ultima versione del software della BridgeBoard. Da quando è stato prodotto il SideCar, infatti, vi sono state varie aggiunte di opzioni e di utility, per non parlare poi delle diverse release del software del SideCar stesso; la cosa più importante, però, è che l'ultima versione del software della BridgeBoard funziona perfettamente anche col SideCar ed è quindi quella da preferirsi.

Il resto di questo articolo si fonda su due ipotesi di base. La prima è che abbiate seguito fedelmente le istruzioni di installazione presenti nel vostro manuale della BridgeBoard e che la BridgeBoard stessa sia stata provata e funzioni perfettamente. La seconda è che abbiate una sufficiente conoscenza sia di MS-DOS sia di AmigaDOS. Troverete utile avere a portata di mano il manuale della BridgeBoard 2088, il manuale del sistema operativo 3.2, il vostro manuale di Amiga, un disco da 3.5" vuoto e alcune ore di tempo libero per fare un po' di prove.

La potenza che scaturisce dall'unione dei due sistemi operativi genera inevitabilmente alcuni problemi e alcune domande. Il fine di questo articolo è di chiarire quegli aspetti che risultano generalmente poco compresi e confusi. Tra le altre cose vedremo infatti come trasferire i file da MS-DOS ad AmigaDOS e viceversa, come approntare un hard disk con la Janus e come far formattare un disco MS-DOS da 720K all'A1010 (il drive da 3.5").

720K sul 3.5"

Incominciamo dalla fine: ottenere 720K sul 1010. Quando connettete il 1010 alla BridgeBoard, questo, per default, formatta i dischi a 360K. Perché ci interessa tanto formattare a 720K? In primo luogo, questo è standard tanto quanto 360K sui dischi da 5.25" sotto MS-DOS. La maggior parte dei portatili MS-DOS usa questo formato e i nuovi modelli IBM sono dotati di drive in grado di leggere e scrivere in questo formato; noi certamente non vogliamo essere da meno. Se poi tutti questi discorsi di compatibilità non vi interessano, basta che osserviate che 720 è il doppio di 360 e questo dovrebbe proprio convincervi.

Bando alle chiacchiere e iniziamo a vedere che cosa bisogna fare per raddoppiare la capacità dei nostri dischi e diventare compatibili. Tutto ciò che dobbiamo fare è aggiungere una linea al file CONFIG.SYS. Il file CONFIG.SYS deve essere contenuto nella directory principale del disco che usate per fare il boot della BridgeBoard. Questo significa il disco di boot in MS-DOS e non il disco che usate per far partire il software che gestisce la finestra PC. Se sul vostro disco di boot MS-DOS non c'è già il file CONFIG.SYS, potete crearlo voi con l'editor di linea "edlin", che vi è stato fornito con la BridgeBoard. Potete usare "edlin", poi, anche se CONFIG.SYS esiste già e se, quindi, dovete solo aggiungere una nuova linea. Le istruzioni d'uso di "edlin" sono riportate nel Capitolo 6, seconda parte, del manuale del sistema operativo. Ecco la linea che dobbiamo aggiungere:

```
driveparm=/D:01 /F:2 /H:2 /S:9 /T:80
```

Gli spazi fra i diversi termini sono necessari; è fondamentale che mettiate esattamente questa linea così come è riportata qui. Il significato di questa linea e dei vari parametri è spiegato nell'Appendice B del manuale MS-DOS 3.2. Ora dobbiamo rendere attivo questo nuovo file CONFIG.SYS. A questo scopo dobbiamo operare il cosiddetto *warmstart* della BridgeBoard, cioè dobbiamo farla ripartire. Per fare il reset della sola BridgeBoard, *clickate* con il mouse dentro la finestra PC per renderla attiva e quindi premete ALT, CTRL e DEL (il punto nel tastierino numerico). A questo punto il vostro sistema vedrà il drive B come drive da 3.5" a 720K. Per verificare che tutto sia andato come previsto, basta semplicemente formattare un disco nel

drive "B". Al termine di tale operazione dovrebbe apparirvi questa scritta sullo schermo:

```
730112 Bytes total disk space
730112 Bytes available on disk
```

Se così non fosse, ricontrollate il file CONFIG.SYS.

Nota: questo trucco funziona anche se avete due drive da 5.25" (A1020) sulla BridgeBoard; il problema è però che i dischi da 5.25" formattati a 720K non possono essere letti dalla maggioranza degli altri utenti MS-DOS.

Drive virtuali

Il secondo argomento di cui ci occuperemo è costituito dai drive virtuali. I drive virtuali sono trattati nel manuale della BridgeBoard, ma richiedono comunque qualche chiarimento e qualche ulteriore spiegazione. Io vi fornirò un esempio di come approntare un drive virtuale, ma è lo stesso utile che troviate il tempo di leggere l'Appendice C del manuale della BridgeBoard, se non lo avete già fatto. Per usare i drive virtuali bisogna compiere tre operazioni. Il primo passo consiste nell'aggiungere la seguente linea al file CONFIG.SYS:

```
device=jdisk.sys
```

Una volta che avete salvato il file con questa modifica, potete *resettare* la BridgeBoard e passare alla seconda fase. Bisogna notare che mentre la prima operazione viene eseguita una volta per tutte, le due successive devono essere compiute ogni qual volta si intenda usare un drive virtuale.

Il secondo passo consiste nel lanciare il programma "PCdisk", che si trova nel drawer PC nello schermo del Workbench di Amiga. A questo punto abbiamo creato un canale di comunicazione tra MS-DOS e AmigaDOS. In questo esempio, useremo il disco RAM di AmigaDOS per ospitare il drive virtuale, ma potremmo allo stesso modo usare un qualsiasi altro drive.

Come terza operazione, creiamo il drive virtuale. Battete nello schermo MS- DOS il comando seguente:

```
jlink d: ram:ms-dos /c:160
```

e premete Return. Come risultato, sullo schermo dovrebbe apparire:

```
VDrive   Status   Linked to
---------------------------
c:
d:       R/W      ms-dos
e:
f:
```

In questa maniera abbiamo creato un file chiamato "ms-dos" nel RAM disk di Amiga. Questo file è collegato alla parte MS-DOS tramite il drive virtuale D. La sua presenza nel RAM disk di Amiga può essere verificata con la richiesta di una directory di RAM: (ovviamente da AmigaDOS); è importante precisare che questo file non può essere letto, copiato o cancellato da AmigaDOS fino a quando non viene chiuso (e vedremo tra poco come farlo). Ad ogni modo, ora possiamo accedere al drive D come a un qualsiasi altro drive MS-DOS.

Per provarlo, potete ad esempio chiederne la directory, copiarvi il file CONFIG.SYS e leggerlo con il comando TYPE, cancellarlo e chiedere un'altra volta la directory. Se non sapete come chiedere la directory, o come copiare un file, è opportuno che leggiate il manuale MS-DOS 3.2.

Ora che disponiamo di un drive virtuale e sappiamo usarlo, spiegherò il significato dei parametri specificati col comando jlink. La "d:" indica che intendiamo chiamare il drive virtuale con il nome D. "ram:ms-dos" specifica che vogliamo che il drive virtuale sia contenuto nel file chiamato "ms-dos" nel device RAM: di Amiga. L'opzione "/c:160" serve per limitare la lunghezza massima di tale file a 160K. Se non usiamo l'opzione "/c" e il file specificato non esiste già, il comando jlink non ha alcun effetto e viene restituito un messaggio di errore. Ho usato di proposito il numero 160 perché questa è la dimensione minima che possa essere specificata per un drive virtuale (pensavate che me lo fossi inventato, vero?). Se usate un numero minore di 160, la dimensione del file è posta automaticamente a 160K. In effetti, 160K non è la vera e propria dimensione del file, ma è la massima dimensione che può raggiungere. A giudicare dall'espressione dei volti, direi che è meglio che spieghi più chiaramente questo punto...

Un drive virtuale funziona sostanzialmente come il RAM disk di Amiga nel senso che non occupa mai più spazio di quello che effettivamente gli serve (a meno che non si superi la dimensione specificata nel comando jlink). Questo sta a significare che il drive virtuale si espande solo quando vi salviamo dei file; a differenza, però, del RAM disk di Amiga, quando cancelliamo i file, il drive non si restringe, ma continua ad occupare lo stesso spazio. Un drive virtuale si espande solamente e non si restringe mai.

È ora opportuno parlare di due aspetti importanti: come sconnettere e riconnettere un drive virtuale. Un drive virtuale dovrebbe sempre essere scollegato al termine del suo uso e prima di spegnere la macchina. Se spegnete tutto senza aver scollegato un drive virtuale, il file che ospitava il drive virtuale potrebbe risultare danneggiato e potreste incontrare dei problemi nel riconnettervi a tale file la volta seguente. Vediamo come scollegare un drive virtuale. Battete la linea seguente:

```
jlink d: /u
```

e premete Return. Sul display vi dovrebbe apparire:

```
VDrive   Status   Linked to
---------------------------
c:
d:
e:
f:
```

Questo indica che il nostro drive è stato scollegato e che al momento non ci sono altri drive collegati.

Nota: Vorrei spendere un parola per avvertirvi della pericolosità di collegarsi automaticamente a un drive virtuale durante lo start-up (cioè mettendo un apposito comando nel file AUTOEXEC.BAT). La BridgeBoard inizia la sua attività (a patto che trovi un disco di boot nel drive A) quando viene eseguito il comando "BindDrivers" di Amiga, indipendentemente dal fatto che abbiate aperto la finestra PC o meno. Questo significa che se collegate il drive virtuale automaticamente all'accensione e poi spegnete senza aver aperto la finestra PC e scollegato il drive virtuale, finirete molto probabilmente col danneggiare il file che ospita il drive virtuale. Ci sono tre modi per impedire il verificarsi di questo incidente. Il primo è di non usare il comando jlink nel file batch di start-up. Il secondo consiste nell'aprire sempre e comunque una finestra PC per scollegarsi prima di spegnere, il che è più facile da dire che da ricordare. Il terzo, che personalmente preferisco, è di inserire il comando "date" (o eventualmente "Time") nel file AUTOEXEC.BAT prima del comando jlink. Questo fa sì che se non aprite una finestra PC e non battete la data, non c'è alcun pericolo di perdere il contenuto del drive virtuale semplicemente perché il comando jlink non viene eseguito. Devo poi farvi notare che tutta questa discussione sul collegamento automatico del drive virtuale si basa sull'ipotesi che voi abbiate già lanciato il programma PCdisk manualmente o come parte della "startup-sequence".

Ricollegarsi al file che ospita il drive virtuale è semplice; come esempio, riconnettiamoci al drive virtuale che abbiamo usato per le nostre prove. Battete questa linea:

```
jlink d: ram:ms-dos
```

e premete Return. Vi dovrebbe apparire come risposta:

```
VDrive    Status    Linked to
---------------------------
c:
d:        R/W       ms-dos
e:
f:
```

Come potete vedere, il nostro drive virtuale è di nuovo pronto per l'uso. E' importante che notiate che non ho usato l'opzione "/c" questa volta; se lo avessi fatto, avrei perso tutti i dati contenuti nel vecchio file.

Se usate il comando jlink senza parametri, vi verrà mostrata la lista dei nomi di drive disponibili e lo stato di tutti quelli attivi. Forse avete già notato che jlink produce una lista di quattro nomi di drive e questo avviene semplicemente perchè sono quattro i drive virtuali diversi che il sistema vi consente di usare. Si può usare uno qualsiasi dei nomi specificati senza dover rispettare un ordine specifico. Nel caso che abbiate più o meno di due drive, la lista inizierà con nomi diversi; infatti la lista inizia col primo nome di drive libero.

Lasciatemi concludere l'argomento dei drive virtuali con un accenno all'opzione "/r". Questa opzione, se usata alla creazione o alla riconnessione, fa in modo che il drive virtuale sia di sola lettura. Tutti i tentativi scrittura su questo drive falliranno e saranno accompagnati da un messaggio di avvertimento.

Congratulazioni! Siete sopravvissuti al corso introduttivo sui drive virtuali. Passiamo allora senza indugi al prossimo argomento.

I drive Janus

Un drive Janus è in realtà una partizione AmigaDOS che risiede su un hard disk MS-DOS e a cui AmigaDOS accede come a un qualunque device. Grazie al fatto che MS-DOS permette che su un disco siano presenti diverse partizioni e grazie ad alcune utility messe a disposizione dalla Commodore, possiamo trasformare una di queste partizioni in un drive Janus.

A causa del numero enorme di differenti tipi di drive e controller di hard disk, non mi sembra opportuno che spieghi come installare un hard disk. Nel nostro caso ci riferiremo a un hard disk da 20M partizionato in parti di uguale dimensione; in generale, potrà rendersi necessario fare alcuni specifici aggiustamenti delle dimensioni delle partizioni.

La BridgeBoard, come qualsiasi altro computer che usi MS-DOS come sistema operativo, richiede che, prima di formattare un hard disk, venga lanciato il programma "Fdisk", che potete trovare nel disco di sistema fornito con la BridgeBoard. La funzione di "Fdisk" è spiegata in grande dettaglio nella Parte 3 - Appendice F del manuale MS-DOS 3.2; per questa ragione, piuttosto che riscrivere delle informazioni che potete benissimo trovare da altre parti, preferisco mettervi solamente in grado di usare Fdisk per creare la partizione per il drive Janus e rimandarvi al manuale MS-DOS per ulteriori chiarimenti.

Battete "Fdisk" e premete Return. Una volta che il menu è apparso, scegliete l'opzione 1; dal momento che intendiamo usare parte di questo drive come drive Janus, battete "n" quando vi viene chiesto se volete usare l'intero hard disk. A questo punto ci viene mostrato il numero complessivo dei cilindri disponibili per l'uso (611) e siamo invitati a precisare la dimensione della nostra partizione. Poiché vogliamo dividere il drive a metà, battete 312 e premete Return. Ci viene poi richiesto il numero del primo cilindro da assegnare alla partizione che stiamo creando; battete 0 e premete Return. A questo punto Fdisk ci informerà che la partizione MS-DOS è stata finalmente creata. Selezionate allora l'opzione 2; Fdisk vi chiederà di inserire il numero della partizione che intendiamo rendere attiva. Battete "1" e premete Return. Con questo abbiamo terminato di usare Fdisk e per uscire possiamo usare il tasto Esc; una volta usciti, è necessario fare nuovamente il boot della BridgeBoard.

Compiuta questa operazione, accingiamoci a creare una partizione per il drive Janus. Battete "Adisk" e premete Return; vi dovrebbe comparire una serie di messaggi e di opzioni simili a quelle presenti in Fdisk. I messaggi ci informano che al momento non ci sono partizioni gestite da Amiga, che il numero complessivo di cilindri disponibili è 611 e che c'è una partizione MS-DOS attiva. Questa partizione MS-DOS ha inizio col cilindro 0 e termina al cilindro 311. Scegliete l'opzione 3. Ci

viene allora comunicato il numero di cilindri disponibili (298) e il numero del primo cilindro libero (312). Seguendo le indicazioni, battiamo 298 e 312; fatto questo, Adisk ci informerà che ora la partizione numero due è una partizione Amiga con inizio al cilindro 312 e termine al cilindro 611. La divisione nelle due partizioni da 311 e 298 cilindri non è proprio a metà esatta, ma gli si avvicina sufficientemente. Anche in questo caso, usate il tasto Esc per uscire e date subito il reset della BridgeBoard.

Il passo successivo consiste nella formattazione delle due partizioni del nostro hard disk. Per far questo, aprite la finestra PC e battete il comando seguente:

```
format c: /v /s
```

e premete Return. Il sistema correttamente ci avverte che tutti i dati presenti sul disco "non-removable", cioè sullo hard disk, verranno cancellati. Premete "y" e quindi Return per proseguire nell'operazione. A questo punto viene formattata la sola partizione MS-DOS; quando l'operazione è terminata, ci viene richiesto un nome per il volume creato. Battete il nome che avete scelto e premete Return. La partizione MS-DOS è ora pronta per l'uso.

Passiamo ora dalla parte AmigaDOS e apriamo una finestra CLI. Se non avete familiarità con il CLI, è consigliabile che leggiate il vostro manuale di Amiga. Battete "DJmount" e premete Return. Dopo pochi secondi dovrebbe apparire un requester che vi informa che il vostro hard disk non è un disco DOS. Ignorate del tutto questo messaggio e scegliete col mouse "Cancel". Battete la linea seguente:

```
DPformat drive jh0: name "Nome scelto"
```

dove al posto di "Nome scelto", mettete il nome che avete effettivamente scelto per la partizione AmigaDOS. Dopo che avete premuto Return, apparirà un requester che vi chiede di inserire un disco, ma poichè il disco è già presente, ignorate il messaggio e premete Return. Il sistema formatterà ora la partizione AmigaDOS, ovvero il drive Janus. Al termine della formattazione, ritorna il solito prompt del CLI e nel Workbench appare una nuova icona per il drive Janus appena creato.

Ciò che ora rimane da fare è far sì che il drive Janus si monti automaticamente all'accensione e durante la fase di boot. Dal CLI battete:

```
ed df0:s/startup-sequence
```

Mettete una copia del vostro disco di sistema nel drive df0: e premete Return. Quando la finestra di ED si è aperta, muovete il cursore esattamente alla destra della "s" del comando "BindDrivers". Premendo ora Return si apre una nuova linea; in tale linea battete "DJmount" e premete il tasto Esc. Apparirà un asterisco nell'ultima linea dello schermo; battete "x" e Return.

Avendo inserito il comando DJmount nella startup-sequence, cioè nella sequenza di comandi che vengono eseguiti ogni volta che viene fatto ripartire il sistema, il drive Janus si monterà automaticamente durante la fase di boot. In compenso questo comporta un'attesa di circa un minuto e mezzo ogni volta che viene fatto il boot.

Ciò esaurisce l'argomento dei drive Janus. Giunti a questo punto, dovreste ormai iniziare a sentirvi degli esperti della BridgeBoard!

Il trasferimento di file

L'ultima parte di questo articolo è dedicata a esaminare le diverse maniere per trasferire file tra AmigaDOS e MS-DOS. Ci sono diverse utility già pronte per questo scopo; alcune sono fornite direttamente con Amiga, alcune con la BridgeBoard e altre sono programmi commerciali disponibili presso i rivenditori.

È importante tenere a mente che poter trasportare un file binario ovvero un programma da un sistema all'altro non significa assolutamente che quel programma funzioni, anzi questo non capita mai. I trasferimenti di cui ci occupiamo non comportano alcun tipo di conversione e in generale hanno senso solo per file di testo.

Iniziamo ad esaminare le PCutils. Queste utility sono state prodotte dalla Commodore e inserite nell'Enhancer Kit della versione 1.2 di AmigaDOS; attualmente vengono fornite come corredo software con ogni Amiga. Le PCutils consistono in due programmi, PC Format e PC Copy, e funzionano con dischi da 5.25" formattati sotto MS-DOS a 360K e solo con il drive A1020. L'uso di PCutils non consente che l'A1020 venga montato, cioè reso disponibile per l'uso con AmigaDOS.

PC Format permette di formattare un disco MS-DOS col drive A1020 e risulta molto comodo perché vi evita di ricorrere all'uso della BridgeBoard qualora, ad esempio, vogliate trasferire dei file e vi siete scordati di formattare un disco MS-DOS. L'uso di PC Format è estremamente semplice: dopo aver attivato col mouse l'icona di PC Format, vi appare una finestra con i valori di default delle opzioni. Potete cambiare a vostro piacimento le opzioni e inserire il nome di volume che desiderate; in generale, comunque, non avrete alcun bisogno di alterare i valori di default.

PC Copy è il programma più usato dei due che compongono le PCutils. PC Copy può copiare sia file di testo sia file binari. Ci sono due opzioni specifiche per i file di testo; Text-7 e Text-8 sono due "filtri gadget" che assicurano che i file copiati siano convertiti correttamente per l'impiego con l'altro sistema operativo. PC Copy può infatti essere usato per leggere e scrivere tanto i dischi MS-DOS quanto quelli AmigaDOS.

Quando viene lanciato PC Copy, le opzioni sono predisposte per la copia di file binari dal 1020 al formato AmigaDOS. Se intendete copiare file di testo, dovete cambiare oppotunamente l'opzione di filtraggio. Se quando avete lanciato PC Copy non avevate posto un disco nel 1020 o se nel frattempo avete cambiato il disco, clickate sulle parole "Disk Change" nell'angolo in basso a sinistra per informarne PC Copy. Se il disco MS-DOS che state usando contiene file nascosti, clickate sulla pa-

rola "Hidden" al centro nella parte bassa dello schermo e tali file vi verranno mostrati. Se desiderate copiare da AmigaDOS a MS-DOS, clickate all'interno della finestra "From: - To:". Questo scambierà il drive di destinazione con quello di sorgente. Quando copiate da AmigaDOS a MS-DOS avete poi tre ulteriori opzioni, la cui attivazione è controllata da un menu "pull-down". Queste opzioni sono: Read-Only, Hidden e System. Se non avete idea di che cosa queste parole significhino, potete trovare tutte le spiegazioni del caso, come al solito, sul manuale MS-DOS 3.2. Questo è quanto è utile sapere sulle PCutils, per cui se avete un drive A1020, provate a dar loro un'occhiata; non saranno proprio le migliori, ma dal momento che sono praticamente gratis, hanno un ottimo rapporto prezzo-prestazioni.

Passiamo ora a considerare le utility ARead e AWrite fornite sul disco MS-DOS con la BridgeBoard. E' bene sottolineare che affinché ARead e AWrite possano funzionare bisogna prima far eseguire il comando PCdisk, che ho già menzionato nel paragrafo relativo ai drive virtuali.

ARead copia un file AmigaDOS in un file MS-DOS. Il file AmigaDOS può essere contenuto in un qualsiasi device riconosciuto da AmigaDOS e quindi sui floppy disk, sugli hard disk, nel RAM disk e così via. Il file destinazione può invece risiedere in un qualsiasi device MS-DOS. Il seguente esempio chiarisce la sintassi del comando:

```
ARead df0:s/startup-sequence A:test.txt
```

In questo caso, il comando ha come effetto quello di copiare il file di Amiga "startup-sequence" dalla directory "s" del drive "df0:" nel drive MS-DOS "A:" con il nome di "test.txt". A questo riguardo, è opportuno far notare che la Commodore si è dimenticata nel manuale della BridgeBoard di precisare che nel caso in cui vogliate copiare un file binario, dovete aggiungere l'opzione "/B" alla fine della linea del comando. A puro titolo di esempio, per copiare il file eseguibile "echo" (che non funziona su MS-DOS, ovviamente!) si deve dare il seguente comando:

```
ARead df0:c/echo A:echo.exe /B
```

AWrite fa l'opposto di ARead; trasferisce infatti un file dall'ambiente MS-DOS all'ambiente AmigaDOS. Per AWrite valgono le stesse considerazioni fatte per ARead. Un esempio del suo impiego è il seguente:

```
AWrite A:autoexec.bat df0:testfile
```

Il risultato è che il file "autoexec.bat" contenuto nel drive "A:" viene trasferito in ambiente AmigaDOS sul drive "df0:" col nome di "testfile". Anche nel caso di AWrite bisogna usare l'opzione "/B" quando si copiano file binari.

Un impiego interessante di AWrite consiste nel trasferire file di testo da MS-DOS alla stampante di Amiga. La sintassi è questa:

```
AWrite A:autoexec.bat prt:
```

Il vantaggio di usare AWrite al posto del programma "LPT1" sta nel fatto che in questa maniera si può usare Preferences per controllare il tipo di stampa o per usare una qualunque delle stampanti che Amiga può pilotare.

Diamo ora un'occhiata a Dos-2-Dos. Dos-2-Dos è un programma commerciale che, come PC Copy, gira sotto AmigaDOS e può impiegare il drive 1020. Le somiglianze, però, si fermano qui. Dos-2-Dos, infatti, può leggere e scrivere dischi MS-DOS sia da 5.25" che da 3.5", da 360K o 720K.

Dos-2-Dos è molto semplice da usare e ha il solo difetto di partire solo da CLI. Non intendo spiegare qui come usare Dos-2-Dos, dal momento che ciò è ampiamente spiegato nel manuale che accompagna il programma. I vantaggi principali di Dos-2-Dos rispetto a PC Copy sono la possibilità di usare qualsiasi drive, anche se già montato, e la possibilità di leggere e scrivere i dischi MS-DOS da 3.5". Queste due peculiarità possono giustificare senz'altro l'acquisto del programma.

Nota: Dos-2-Dos fornisce anche un'altra curiosa possibilità: quella di leggere e scrivere anche dischi formattati con l'Atari ST.

L'ultimo programma che intendo menzionare in questa breve rassegna dei programmi di trasferimento file si chiama "Project D". Project D è una utility di backup per Amiga con un nuovo accessorio chiamato Omni Copy. Omni Copy è diverso dalle utility di cui ho parlato in precedenza; in particolare, non copia file da un sistema operativo all'altro. Omni Copy, infatti, si limita a copiare dischi del medesimo DOS: anche su formati diversi e con più drive destinazione.

Per rendersi conto delle possibilità che Omni Copy offre, considerate la seguente situazione: voi avete un Amiga con un drive 1020 e un vostro amico ha un 500 con il Transformer. Questo vostro amico vorrebbe far andare il Transformer sul suo 500, ma non ha un disco di boot MS-DOS da 3.5". D'altro canto, possiede un PC compatibile e quindi ha sicuramente un disco di boot MS-DOS da 5.25". Ciò che il vostro amico deve fare è portarvi il disco di boot da 5.25" e un disco da 3.5" vuoto; con Omni Copy, infatti, potete trasferire il contenuto del disco MS-DOS da 5.25" nel disco da 3.5", sempre in formato MS-DOS.

Ovviamente Omni Copy copia anche da 3.5" a 3.5" e da 3.5" a 5.25"; con un solo programma, quindi, potete trasferire liberamente il contenuto dei dischi di formato diverso. Considerando, poi, che Project D è venduto sostanzialmente come utility di backup per Amiga, le possibilità offerte in aggiunta e la versatilità di Omni Copy non possono che rendere questo prodotto molto interessante.

Sani e salvi alla meta

Siete riusciti ad arrivare al termine della guida all'esplorazione della BridgeBoard. Spero che la conoscenza che avete acquisito in questo viaggio si dimostri utile per evitare eventuali problemi e che serva per farvi apprezzare pienamente le qualità della BridgeBoard.

ViewPort

La vita, l'universo...e la legge di Sturgeon

di Larry Phillips

Larry Phillips, di Vancouver, British Columbia, si occupa di hardware e software per passione. È anche SysOp nei CompuServe's Amiga Forum.

Qualcuno una volta ha detto all'autore Theodore Sturgeon: "il novanta per cento della scienza è costituito da fesserie". La risposta di Sturgeon ha colpito nel segno. Disse: "il novanta per cento di tutto è costituito da fesserie".

Questa piccola raffinatezza fin da allora è conosciuta come la Legge di Sturgeon, e a seconda del vostro punto di vista può essere più o meno vera. La cosa importante è che è il vostro punto di vista che determina la veridicità o la falsità della Legge di Sturgeon poiché si adatta a qualsiasi cosa la applichiate.

Recentemente ho avuto l'occasione di rispondere a un commento su di un prodotto che presto sarà in vendita per l'Amiga. Ero completamente contrario a tale commento. Prima di esporre il commento o le ragioni del mio disaccordo voglio dare alcune informazioni precedenti.

Gli interpreti PostScript vengono di solito inseriti nella stampante, come ad esempio la NEC LC-890 o la LaserWriter della Apple, oppure in una compositrice come la Linotron. Il codice PostScript viene mandato alla stampante e viene avviato attraverso l'interprete che si trova all'interno della stessa. Questo produce un output alla risoluzione del dispositivo di output, che può andare da 300 DPI (Dots Per Inch/Punti per pollice) per le stampanti laser, a 1200 DPI o 2540 DPI per la Linotron.

Il PostScript permette la definizione di oggetti che sono indipendenti dalla risoluzione della periferica. Questi possono essere oggetti geometrici come cerchi, quadrati, linee, o una combinazione degli stessi. Si può trattare di caratteri individuali in un font particolare di tutte le dimensioni che volete.

A causa della natura indipendente della risoluzione PostScript, l'output di una periferica che può lavorare in PostScript apparirà della miglior qualità possibile per quella periferica. Una stampante laser a 300 DPI non produrrà un output buono come quello della Linotron a 1200 DPI, e l'output della Linotron a 2540 DPI sarà ancora migliore. Questa indipendenza nella risoluzione è sfruttata dai compositori e dal settore dell'editoria.

Date un'occhiata alla pagina che state leggendo in questo momento. Notate la qualità dei caratteri, i bordi, i diversi font usati. Questa pagina è stata prodotta in PostScript, tutta, compresi i bordi.

Molte editorie non possiedono una Linotron. Sono macchine molto costose (circa 100 milioni - nde), ed inoltre non sono necessarie allo scopo di comporre libri o riviste, quindi la spesa sarebbe inutile.

Le editorie potrebbero utilizzare una stampante laser in grado di lavorare in PostScript collegata ad un Amiga. La risoluzione a 300 DPI è sufficientemente adatta a molti scopi. Sicuramente è in grado di dire all'editore come apparirà una pagina; si può vedere in anticipo il prodotto della pagina e quindi sapere che quando appare corretta, apparirà nello stesso modo su di una compositrice più costosa.

I compositori sono persone molto pignole quando si parla in termini di qualità di stampa. Sono in grado di dire quasi immediatamente quale risoluzione è stata usata per produrre un libro o una rivista. Vi diranno anche che una risoluzione a 300 DPI non è sufficiente e che la risoluzione a 1200 DPI è l'unica possibile per avere determinati risultati.

Avete mai osservato l'output di una stampante laser che lavora in PostScript? È probabile che abbiate pensato che fosse un risultato maledettamente buono, forse persino eccellente, ma solo se non siete un compositore.

Tutto dipende dal vostro punto di vista. Dopo avervi dato sufficienti informazioni anche voi potreste osservare che una stampante a 300 DPI è veramente inutile.

Ritornando al commento, una società chiamata Pixelations spera di mettere sul mercato un prodotto nel prossimo febbraio, chiamato PrintScript, un interprete PostScript, ma con una differenza. Eseguirà tutto il lavoro sulla memoria del vostro Amiga invece che nella memoria di una costosa stampante laser, inoltre eseguirà il lavoro alla risoluzione di cui avete bisogno per la vostra stampante, il che significa che potete dare l'output in PostScript alla vostra stampante, che normalmente non è in grado di lavorare in PostScript, la quale può essere a 9 aghi, 24

aghi, a getto d'inchiostro o persino laser.

In pratica prenderà il codice PostScript e costruirà una bitmap che coincida con le capacità della vostra stampante, poi invierà il codice alla stampante come un dump grafico. Questo è esattamente ciò che fà una stampante PostScript, ma una stampante ad aghi sarà molto più lenta ed avrà una risoluzione inferiore, sebbene alcune stampanti ad aghi possano avere una risoluzione di 360 DPI; tuttavia i loro punti sono più sfuocati.

Lo stesso codice PostScript può essere creato manualmente, con un editor, oppure con un programma come Professional Page. Il costo di PrintScript sarà di $89, e non si trova perciò nella fascia dei prezzi che voi attribuireste solitamente al software professionale.

Il commento si basava sul fatto che inizialmente PrintScript sarebbe uscito sul mercato con due soli font, molto simili al Times e all'Helvetica, due dei font più popolari disponibili sulle stampanti PostScript.

La persona che ha fatto il commento è un professionista, probabilmente un compositore, si riferiva al fatto che "tutti perdevano tempo attorno ad un programma PostScript che non aveva come supporto una vasta scala di font standard". Dal suo punto di vista, quello cioè di un professionista, ha perfettamente ragione.

Ma qual'è invece il punto di vista delle altre persone, del dilettante, delle persone che vogliono stilare bollettini di informazione saltuari, le etichette degli indirizzi o lettere d'affari?

Per me è sempre stato un mistero il fatto che potessero esistere certi prodotti, normalmente molto costosi, fatti invece in modo da poter avere dei prezzi ragionevoli, con meno caratteristiche e di qualità inferiore.

Nessuno penserebbe mai di usare una telecamera della Mattel per registrare un programma televisivo, tuttavia molte di queste telecamere vengono vendute ai bambini. A loro importa il fatto che vi siano lenti di plastica e che la qualità video sia scarsa? No, certamente no. Riprendono i loro amici facendogli fare cose sciocche, generalmente si divertono un mondo e la spesa è ricompensata.

Vi sono molti esempi che potrei citare, dalle economiche macchine fotografiche di plastica agli stereo composti da due registratori, un piatto incorporato, sintonizzatore, amplificatore e casse, tutto per 59 dollari e 95 centesimi.

Per il software è la stessa cosa. PageSetter era un programma interessante poiché era diretto al mercato casalingo. Alcuni pensavano che fosse un programma scarso perché non offriva le caratteristiche e la qualità professionale dell'output, ma io penso che non abbiano colto il punto della questione. Professional Page è il programma che volevano, e non ho mai sentito nessuno lamentarsi di questo, eccetto per il fatto che costa troppo.

Sotto questo aspetto il software è uguale all'hardware. Ci vogliono molti soldi per poter offrire varie caratteristiche e la qualità, e non ci si può aspettare di avere il miglior software a prezzi bassi.

La tecnica è la stessa sia quando si acquistano dei prodotti software che dei prodotti hardware. Decidete che cosa volete che il pacchetto faccia e poi cercate quello che corrisponde alla vostra disponibilità economica.

PrintScript sarà in grado di soddisfare le necessità. Tali necessità riguardano la capacità di produrre il miglior output che la vostra stampante sia in grado di effettuare, e lo farà ad un prezzo ragionevole. In seguito sarà dotato di più font ed inoltre acquisterà senza dubbio altre caratteristiche.

Il fatto che attualmente non sia uno strumento professionale è dovuto alla dichiarazione di un professionista che lo ha ammucchiato insieme a quel novanta per cento di cui parla Theodore Sturgeon, ma per l'utente casalingo o occasionale PrintScript fa parte di quel dieci per cento che non è spazzatura. Vi saranno tuttavia coloro che si aspetteranno di più, i quali si lamenteranno della perdita di tempo.

L'unico problema riguardante la questione del punto di vista, nasce per quei prodotti che mirano al settore professionale e che sono pubblicizzati come tali, mentre si tratta di prodotti destinati poco più che al mercato casalingo.

Non mi importa se la qualità del mio programma CAD non è a livello dei programmi professionali perché soddisfa i miei bisogni limitati e costa molto meno dei programmi a livello professionale.

Se fosse stato pubblicizzato come programma a livello professionale probabilmente ne sarei rimasto deluso, e l'acquirente che avesse avuto bisogno di un programma a livello professionale certamente avrebbe fatto lo stesso, qualsiasi fosse stato il prezzo.

Penso che se una società si consultasse con dei professionisti e si preoccupasse di descrivere il suo prodotto accuratamente, piuttosto che rischiare l'ira di coloro che richiedono le caratteristiche descritte in maniera pomposa e risplendente sul prodotto, porterebbe alla stessa società grossi vantaggi.

In conclusione, cosa possiamo dire dell'applicazione della Legge di Sturgeon all'attuale linea di prodotti software per Amiga? Beh, se avete necessità professionali la Legge di Sturgeon è vicina alla verità, sebbene sia un po' conservatrice. Se invece fate parte degli utenti casalinghi medi allora il novanta per cento va probabilmente ridotto di uno o due fattori.

Se siete sviluppatori di software, perché non passare in rivista attentamente i vostri prodotti, con l'aiuto di un professionista nel settore adeguato, e chiamarli con il loro vero nome?

Non abbiate paura di fornire un prodotto diretto al mercato casalingo, semplicemente chiamatelo come tale, ed usatelo come base per uno sviluppo futuro.

Notizie Amiga

MS-DOS e FFS rivisitati, e un'occhiata al software per l'A500

di Don Curtis

Don Curtis è un funzionario di polizia di Denver, in Colorado. Negli ultimi due anni è stato assegnato alla progettazione e allo sviluppo di programmi oltre che alla progettazione ed alla manutenzione di sistema di 10 computer AT&T Unix 3B2. Durante il tempo libero Don è assistente Sysop nei CompuServe's Amiga Forum.

Come ormai avrete notato, l'ultima volta vi ho dato un'informazione errata. Ho descritto un metodo con il quale avreste potuto usare il nuovo Fast Filing System (FFS) su di una partizione MS-DOS dell'Hard Disk per Amiga (drive JANUS).

Come vi ho detto nel precedente articolo, non avevo verificato tutte le procedure da me descritte, ma pensavo provenissero da fonti attendibili. In questo caso particolare l'informazione proveniva direttamente da uno dei programmatori software della Commodore. Per dirla in parole povere: si sbagliava.

Ho un A2000 fornito di Bridge Board e di hard disk (10 Megabyte) sulla partizione MS-DOS, ma ho anche un hard disk sulla partizione Amiga quindi non ho bisogno di usare un drive JANUS. Oltre tutto utilizzo tutti i 10 Megabyte per la parte MS-DOS. Dunque, per farla breve, qualcuno mi ha riferito di aver provato ad usare FFS su di un hard drive JANUS e che il drive non riusciva ad eseguire il format. Il sintomo riportato riguardava il programma format, il quale iniziava nella maniera corretta ed in seguito si fermava semplicemente dopo aver formattato il primo cilindro. Niente, nessun messaggio di errore, nessuna attività, solo un programma bloccato.

Ho pensato che ci dovessero essere dei problemi nella sua mountlist o nella procedura usata, quindi ho fatto un backup del mio drive MS-DOS ed ho provato ad avviare il FFS su di un drive JANUS.

Alcuni problemi: la formattazione è iniziata ma poi si è bloccata. Ho controllato, cambiato i parametri nella mountlist ed ho utilizzato un editor di settore per cercare di capire cosa stesse effettivamente succedendo nel drive. Quella sera devo aver avviato FDISK e ADISK, riavviato la macchina, effettuato il mount e così via per almeno trenta volte, cercando di farlo funzionare. Sapevo che la procedura che avevo dato era vicina a quella corretta, il drive stava effettuando il giusto tipo di attività ed il primo cilindro risultava in realtà formattato, ma niente di più.

Ho notato un'unica stranezza, il normale programma format, DPFormat, cominciava sempre dal cilindro 0 invece che dal cilindro 105, il cilindro dal quale effettivamente partiva la partizione JANUS. Naturamente DJMount e DPFormat funzionavano alla perfezione, ma non era quello che volevo.

Quello era solo l'indizio numero uno per riuscire a far funzionare il FFS. Mentre potevo avere accesso al drive, se davo l'ordine di effettuare un mount sullo stesso il mio editor di settore non mostrava le dimensioni del drive e il numero dei settori. Continuando a sbizzarrirmi con l'editor di settore e con degli ordini nella mountlist, sembrava che, per l'Amiga, il drive JANUS fosse un drive completamente separato, non una partizione su un drive.

Cambiando i numeri dei cilindri nella mia mountlist, in modo che combaciassero con quelli dati da DPFormat una volta avviato, ha risolto quel problema. Ora, se ho usato DJMount per effettuare il mount del drive e DPFormat per formattarlo, avrei potuto semplicemente usare il comando normale Amiga MOUNT invece che DJMount, pur utilizzando il drive. Sfortunatamente non mi ha aiutato a risolvere il problema del format. Continua a bloccarsi.

L'indizio numero due è stato il modo in cui il filesystem identifica un disco. La differenza tra l'Old File System (OFS) e il FFS è il modo in cui i settori dei dati vengono utilizzati. L'OFS usa solo 488 byte dei 512 disponibili per i dati, gli altri 24 byte riguardano le informazioni del filesystem, principalmente quelle sull'eventuale recupero.

Il FFS rinuncia al recupero dati ma utilizza tutti i 512 byte per i dati. Tuttavia le intestazioni della root directory, delle directory e dei file sono identiche sia nell'OFS che nel FFS.

L'AmigaDOS usa un marker nel settore 0 del filesystem per indicargli di che tipo di disco si tratta. Un disco KickStart utilizza i caratteri ASCII KICK come identificazione. Un disco OFS usa i caratteri ASCII DOS, oltre ad un byte contenente il valore 0 come suo segno di identificazione (DOS0). Un disco

FFS utilizza gli stessi caratteri DOS oltre ad un byte contenente il valore1 (DOS1) come suo segno di identificazione. Ricordate che non si tratta delle stringhe "DOS0" e "DOS1" ma delle stringhe "DOS" seguite dal valore 0 o 1.

Adesso, avendo la mountlist corretta, se eseguivo il mount sul drive come drive FFS, ottenevo il requester "Not a Dos Disk". Se invece eseguivo il mount sul drive come drive OFS, funzionava alla perfezione. Entrambi i filesystem riconoscevano il disco, il programma format non funzionava sia usando OFS che FFS ma, una volta formattato, OFS funzionava perfettamente. Che cosa sarebbe accaduto se avessi cambiato il segno di identificazione del FFS? FFS avrebbe potuto in questo caso usare il disco? Certamente! Ho preso quindi il mio editor di settore ed ho cambiato il quarto byte da valore 0 a valore 1, quindi l'ho trascritto. Ho riavviato la macchina, effettuato il MOUNT sul drive con una mountlist FFS e l'ho usato con FFS! Comunque l'AmigaDOS apparentemente usa solo quei primi quattro byte, quindi se avete un editor di settore che richiede che voi eseguiate un checksum sul blocco prima di poterci scrivere, funzionerà sempre.

Questo è stato l'ultimo passo. È un rimedio un po' raffazzonato, ma funziona. Dovrete passare attraverso numerosi stadi, ma alla fine ne varrà la pena. L'unico programma di formattazione che sembra funzionare su di un disco JANUS è DPFormat, il quale capisce solo l'OFS. Il FFS capisce solo un tipo di disco DOS1, ma in questo caso l'OFS ed il FFS sono abbastanza simili a quel tipo; formattando sotto OFS e cambiando il disco in disco DOS1 vi permette di utilizzarlo con il FFS.

Per esserne certo ho avviato alcuni testi utilizzando Diskperf, e mi ha dato i seguenti risultati per OFS:

```
File create/delete:
        create 9 files/sec  delete 16 files/sec
Directory scan:   51 entries/sec
Seek/Read test:   49 seek/reads per sec
r/w speed:        buf 512 byte,  rd 20641 byte/sec,
                                 wr 17476 byte/sec
r/w speed:        buf 4096 byte, rd 20480 byte/sec,
                                 wr 17133 byte/sec
r/w speed:        buf 8192 byte, rd 20164 byte/sec,
                                 wr 16487 byte/sec
```

Non cambia molto, e notate che le dimensioni del buffer più alto non variano quasi. Ora proviamo la stessa cosa con FFS:

```
File create/delete:
        create 8 files/sec  delete 12 files/sec
Directory scan:   106 entries/sec
Seek/Read test:   57 seek/read per sec
r/w speed:        buf 512 byte,  rd 23405 byte/sec,
                                 wr 21845 byte/sec
r/w speed:        buf 4096 byte, rd 68985 byte/sec,
                                 wr 55775 byte/sec
r/w speed:        buf 8192 byte, rd 68985 byte/sec,
                                 wr 56987 byte/sec
```

C'è una bella differenza! E questo è stato ottenuto con un vecchio drive da 10 Megabyte, con un tempo medio di seek di 95 millisecondi sul lato MS-DOS. Se voi possedete un drive più veloce (oggi quasi tutti lo sono), suppongo che possiate ottenere cifre ancora più soddisfacenti.

Che ne dite adesso di un vero test? Ho fatto una copia di tutti i file che si trovano sull'hard disk Amiga sul drive JANUS utilizzando il comando COPY con l'opzione ALL. Poiché questa era una partizione di soli 5 mega non sarebbe stata in grado di tenere tutti i file, così l'ho predisposta in modo che potesse stare nel drive JANUS. Usando OFS ho impiegato 8 minuti e 40 secondi a riempire il drive. Usando invece FFS sono bastati 6 minuti e 30 secondi, e poiché il FFS usa tutti i 512 byte di dati per blocco, ha anche copiato 32 file addizionali (5%) nel drive.

Il caricare e lanciare programmi ci mostra che il FFS sul drive JANUS è stato molto più veloce dell'OFS. Ho avviato altri test per assicurarmi che le partizioni del drive Amiga o MS-DOS non avessero problemi l'una con l'altra. In effetti non ne avevano. Gli editor di settore mostravano che, in entrambe le partizioni, tutti i dati si trovavano dove si presupponeva che fossero.

Tuttavia ho scoperto una cosa: non cercate di usare la partizione MS-DOS durante l'attività del disco JANUS, lo mettereste in ginocchio! Lo MS-DOS non riesce semplicemente a comprendere il concetto di fare due cose contemporaneamente, cosa che invece faccio spesso sulla partizione Amiga. Se avete Transformer, il programma di simulazione del software MS-DOS, sapete quanto è lento. Bene, se cercate di usare il MS-DOS ed accedere al drive JANUS allo stesso tempo, sarà dieci volte più lento di Transformer!

Riassumendo: Il FFS funziona su di un drive JANUS, ma non come ho detto che dovrebbe fare nel il mio ultimo articolo. Ricordate, dovete avere un editor di settore. Ecco come usare il FFS su di un drive JANUS (probabilmente vi sono altri modi, solo che non li ho ancora scoperti):

1 Fate un backup di tutto quello che avete sulla partizione MS-DOS e sulla partizione Amiga, se già possedete una partizione JANUS.

2 Se non possedete ancora una partizione JANUS, ripartite la partizione del drive MS-DOS usando i comandi MS-DOS FDISK e ADISK, non dimenticate di assicurarvi che FDISK renda attiva la sua partizione. Dovrete riavviare la macchina dopo aver utilizzato FDISK e ADISK, quindi ci vuole un po' di tempo.

3 Dopo aver avviato l'Amiga usate DJMount per effettuare il mount della partizione Amiga.

4 Usate DPFormat per formattare la partizione Amiga e mentre la formattazione è in corso, scrivetevi i numeri del cilindro basso e di quello alto. Questo è molto importante! DOVETE formattare il drive; anche se il vostro drive JANUS è settato sull'OFS, dovete riformattarlo.

5 Con un editor di settore andate al blocco 0 del drive. Notere-

te che l'identificazione DOS0 si trova nei primi 4 byte. Cambiate il byte 4 da 00 a 01.

6 Adesso create una mountlist per il vostro nuovo drive. Potete chiamarla come volete, anche JH0:. Usate la mountlist di esempio a pagina A-3 dell'"Enhancer Manual", ma cambiate il device con il nome jdisk.device. Cambiate il numero dell'unità in 0 per la prima partizione. Cambiate le Surfaces (superfici) ed i BlocksPerTrack (blocchi per traccia) perché vadano d'accordo con il vostro drive. Cambiate il LowCyl (cilindro basso) in modo che combaci con il numero basso dato da DPFormat e cambiate l'HighCyl (cilindro alto) in modo che corrisponda al numero del cilindro alto dato da DPFormat. Ecco come appariva la mia mountlist:

```
JH0:

    Device = jdisk.device
    Unit = 0
    Flags = 0
    Surfaces = 4
    BlocksPerTrack = 17
    Reserved = 2
    Interleave = 0 /* tenete questo a 0*/
    LowCyl = 0
    HighCyl = 153
    Buffers = 20
    Stacksize = 4000
    GlobVec = 1
    FileSystem = L:FastFileSystem
    DosType = 0x444F5301 /* Notate che questo è "DOS1"
                           in esadecimale */
```

7 Rimuovete dalla startup-sequence tutti i comandi che si riferiscono al drive JANUS, inclusi tutti i comandi DJMount. Non li userete più.

8 Dopo avere salvato la vostra mountlist e la nuova (le nuove) startup-sequence, attendete i soliti 3 o 4 secondi e poi riavviate l'Amiga.

9 Dopo l'avvio dello startup effettuate il mount del drive FFS JANUS usando il comando mount della versione 1.3.

10 Eseguite un test del vostro drive - vedete se riuscite ad accedervi. Se avete fatto tutto secondo le regole dovrebbe funzionare. Se così è, potete importare nuovamente tutti i file che avevate sul drive e poi rielaborare la vostra startup-sequence in mount JH0: (o come l'avete chiamato).

Se non riuscite ad accedere al drive ripercorrete le tappe descritte. Assicuratevi di averle seguite attentamente. Questa procedura con me ha funzionato benissimo e così è stato per altri che l'hanno provata dopo di me. Non ho provato ad usare più di una partizione Amiga, ma non vedo ragione per cui non dovrebbe funzionare con partizioni multiple JANUS. Fate semplicemente la stessa cosa per ogni partizione (cambiate il DOS0 in DOS1 dopo aver effettuato il DPFormat sulla partizione) e poi costituite una sua mountlist. I numeri delle unità aumenteranno per ogni partizione, cioè: la prima partizione è l'unità 0, la seconda è l'unità 1, e così via.

Commenti sulle prospettive del software

Ho dei sentimenti contrastanti riguardo all'attuale stato delle cose quando si tratta della quantità e del tipo di software disponibile per Amiga. Questo include anche i miei pensieri rivolti all'A500, perché i due concetti sono dipendenti l'uno dall'altro. Certamente l'A500 è un'"ottima cosa" in questo momento. È la più popolare delle macchine Amiga e sta vendendo bene. Effettivamente si può considerare come la macchina che ha permesso alla Commodore di portare ancora il pane a casa. L'A1000 vendeva bene, ma è nulla a confronto con l'A500. Anche l'A2000, nelle sue varie forme, sta vendendo bene, ma ancora, non bene come l'A500.

Se le vendite dell'Amiga hanno ormai raggiunto il milione lo dobbiamo all'Amiga 500, ed è una buona cosa. La mia preoccupazione non è per oggi, ma per il futuro. A lungo andare l'A500 potrebbe non essere una "cosa buona".

L'A500 ha aperto il mercato della linea Amiga a persone che godono semplicemente di un'eccellente macchina per giochi a un prezzo ragionevole. Questo mi va bene, quelle persone meritano la migliore macchina che sono in grado di acquistare con i loro soldi, e sembra che l'Amiga riempia questa nicchia di mercato molto bene.

Non è conveniente espandere la memoria dell'A500, sebbene sia possibile, come lo è invece per l'A2000 ed i suoi discendenti. L'A500 diventa ingombrante quando si aggiungono delle parti al corpo centrale. Una scatola che si trova ad un lato del sistema, già composto da unità centrale e tastiera, funziona, ma non è comodo. Non si può spostare la tastiera per trovare una posizione di scrittura più comoda. Non si può sollevare la tastiera e metterla sul grembo per potersi rilassare con le gambe sulla scrivania (la mia posizione di scrittura preferita quando leggo i messaggi da CompuServe).

Tutto questo non riduce le qualità del computer, significa solo che è un computer non adatto per certe cose. A lungo termine significa anche che la maggior parte degli A500 manterrà praticamente la sua configurazione originale, magari con un secondo floppy e un po' più di memoria. Significa anche che i proprietari di A500 non compreranno software che richiede diversi Megabyte di memoria e/o grandi hard disk. Naturalmente vi saranno delle eccezioni. Vi saranno persone che espanderanno il loro A500 a 8 Megabyte di memoria ed aggiungeranno un hard disk da 100 Megabyte, tuttavia non sarà quello il tipico proprietario di A500. Il tipico proprietario di A500 lo utilizzerà per minimi lavori di elaborazione testi, piccoli database casalinghi, alcuni lavori video, e molti giochi.

Per poter vendere al settore professionale abbiamo bisogno di software di alta qualità, e non ci giova certo una reputazione come produttori di macchine per video giochi. Mentre il vendere al settore professionale può non sembrare vantaggioso, in effetti lo è.

Prendiamo come esempio il mio ufficio. Sebbene io sia asse-

gnato all'ufficio dati, dove tutti sviluppiamo programmi a vari livelli, vi sono più computer in ufficio che nelle nostre case. Non sto parlando di computer principali, parlo dei computer da tavolo. Tutti gli undici componenti dell'ufficio hanno sulla loro scrivania un computer, ma solo sette di noi hanno un computer a casa (due dei quali sono Amiga!). Come succede in molte attività, le macchine che noi utilizziamo sono computer MS-DOS. Questo perché il software di cui abbiamo bisogno è disponibile e perché queste macchine non hanno la fama d'essere macchine per video giochi. Sicuramente non sono i computer più adatti al nostro lavoro, ma sappiamo che avranno un valore oggi come pure domani, è molto semplice.

In passato ho posseduto computer che avevano la fama d'essere computer per video giochi e sfortunatamente questo ha fatto in modo che, dopo poco tempo, i soli programmi disponibili fossero i video giochi. Il così detto software serio, per quelle macchine, si era prosciugato. Certamente vi erano dei word processor disponibili, come pure dei database casalinghi e programmi di gestione finanziaria, ma la qualità del software serio disponibile era scarsa, se non nulla, e vi erano solo tonnellate di video giochi.

Molte persone direbbero: "E allora?". Non hanno bisogno di software serio. Probabilmente importa loro ancora di meno il fatto di poter vendere al settore professionale, sono completamente soddisfatte del software disponibile. Ma vi sono anche coloro che notano una grande mancanza nel software Amiga. Io non mi occupo di lavori video, ma noto certamente la necessità di un software per coloro che sono interessati al lavoro video con l'Amiga. Allo stesso modo, io non mi occupo di video giochi, alcune volte gioco ma in generale non ho molto tempo. Questo però non significa che non vi siano altre persone che desiderano giochi migliori e in quantità maggiore.

E qui entra in gioco l'A500. I produttori conoscono la demografia, sanno che se la maggioranza dei computer venduti riguarda i modelli a prezzi bassi, con un'espansione limitata il software da divertimento venderà di più delle altre forme di software.

Ma c'è anche bisogno di software produttivo. Quello di cui noi abbiamo bisogno, in effetti, è un giusto equilibrio nel software tra il divertimento e la produttività. Questo è ciò che manterrà in buona forma le vendite dell'Amiga.

Sfogliando gli ultimi numeri delle riviste Amiga, noto una quantità terrificante di intere pagine dedicate alla pubblicità di video giochi e solo poche pagine di annunci pubblicitari dedicati al software produttivo. Non ho notato un equilibrio, ma per ora siamo già a buon punto.

Quello che voglio dire è che mi piacerebbe poter raggiungere quell'equilibrio. L'Amiga è una macchina molto versatile, può creare grandi giochi e può anche produrre software incredibile, ma il software deve esserci. Sto chiedendo ai produttori e ai programmatori di prendere in considerazione il tipo di software che offrono.

Parte del software così detto produttivo che ho visto sull'Amiga è semplicemente spazzatura. Viene pubblicizzato come software a livello professionale ed è ingombrante, lento e inflessibile. Che vantaggi dà un DataBaseX (ragazzi, spero che non esista un prodotto con tale nome!) se non è neppure in grado di importare un file dBase III? Non dico che debba saper trattare i programmi in dBase, ma nessun rappresentante del settore produttivo si convincerà del fatto che DataBaseX è un programma a livello professionale se non è in grado di accettare i suoi dati senza dover immettere nuovamente 30.000 record a mano.

I word processor che impiegano un'ora a stampare una lettera perché stampano TUTTE le lettere in grafica ad alta risoluzione non sono vantaggiosi. Potrebbe andare bene per stampare lettere occasionali ad uso casalingo, ma in un'attività dove si producono 10 o 20 lettere al giorno non si può aspettare tutto quel tempo. Tali attività hanno bisogno di lettere con un formato appropriato, di un'intestazione e di un testo puliti e precisi. Hanno il denaro per comperare una stampante di qualità che non abbia bisogno di stampare in modo grafico per ottenere testi precisi. Non sto dicendo di non fornire l'output grafico per coloro che ne hanno bisogno, ma non ignorate l'output ASCII per coloro che possono usarlo. Fate in modo che l'utente possa scegliere le sue preferenze, non forzatelo a scegliere le vostre preferenze.

Se avete intenzione di disporre del vostro tempo per creare programmi, dedicatene una buona parte alla cura degli stessi. Un mese in più passato a "levigare" il programma può fare la differenza tra la notte ed il giorno. Prestate ascolto agli utenti, scoprite quello di cui hanno bisogno e dateglielo. Osservate quello che fate voi, quello di cui avete bisogno per fare il vostro lavoro, e cercate di produrre il software per rendere il vostro lavoro, e quello di tutti coloro che ne svolgono uno simile, più semplice.

Se state scrivendo il programma per un video gioco, bene, ma create il miglior video gioco che abbiate mai visto. Poi prendete alcuni dei trucchi che avete utilizzato per far funzionare il gioco ed applicateli ad un programma di applicazione. Non fossilizzatevi, variate la vostra linea di prodotti. A lungo andare, io (l'utente) e l'Amiga saremo tutti vincitori!

Il fatto che l'A500 sia il più popolare della linea Amiga non significa che ne sia l'unico membro. Non ignoratelo, ma non scrivete solo per l'A500. Se esiste il software, anche le macchine più grosse aiuteranno a far salire le vendite.

L'Amiga è una grande macchina per giocare, ma NON è un computer per video giochi. È in grado di fare cose che molti non si sarebbero neppure sognati pochi anni fa. Sfruttiamo queste capacità in ogni maniera possibile.

Linguaggio Assembly

Parte 1 - il nostro primo programma in linguaggio assembly

di Jim Butterfield

Jim Butterfield non ha bisogno di essere presentato. Il suo nome costituisce un punto di riferimento fra gli entusiasti utenti Commodore di tutto il mondo. Ha cominciato a interessarsi di microcomputer ai tempi del KIM-1 da 1K. Jim possiede una conoscenza enciclopedica dei prodotti Commodore, come testimoniato da articoli, libri, lezioni e addirittura da programmi televisivi. In questo articolo Jim presenta la fine arte della programmazione in linguaggio assembly sull'Amiga.

Mi sembra che la programmazione in linguaggio assembly su Amiga sia nascosta alla vista del programmatore principiante medio. È nascosta in molti modi: nei file *include*, nelle macro e nelle definizioni esterne.

Questa situazione è difficile da evitare quando si ha a che fare con programmi di grosse dimensioni. Questo vuol dire, per il principiante, che il programma che vede pubblicato è spesso un piccolo frammento dell'insieme.

Quando si diventa esperti, si cominciano a usare le scorciatoie e le semplificazioni alla stessa maniera. Quel codice di startup tedioso e meticoloso verrà quindi lasciato da parte e verrà incorporato nel programma come materiale già pronto, in scatola. Conosceremo a memoria tre dozzine di definizioni esterne di frequente uso e le chiameremo spensieratamente con il loro nome. Metteremo insieme i nostri pezzi di codice preferiti; per esempio, il codice per stampare un numero decimale ci sarà necessario in continuazione. Scriviamolo una volta, per bene, e incorporiamolo quando sarà necessario nei programmi che svilupperemo.

Ma il principiante si sente come se stesse procedendo nel bel mezzo di una storia senza avere letto i primi capitoli. È difficile sentirsi a proprio agio in una situazione del genere.

Tenterò quindi di procedere lentamente in queste prime sessioni. Più avanti considererete alcune di queste cose tediose, ma ciò non vi impedirà di metterle via come parte della vostra libreria.

Completamente principianti?

Il fatto di arrivare a scrivere dei programmi finiti il più velocemente possibile è una cosa rassicurante. Per arrivare a conseguire questo obiettivo, sarò costretto a trattare speditamente alcune cose che i principianti dovranno approfondire.

Argomenti come le notazioni binaria ed esadecimale, i bus indirizzi e dati, i cicli di lettura ed esecuzione di un'istruzione, le periferiche (devices) di input/output e lo stesso sistema operativo verranno passati in rassegna molto velocemente, come se fossero perfettamente ovvii.

Perdonatemi se vado troppo veloce su queste parti. Esistono libri nei quali vengono trattati in modo specifico alcuni di questi concetti e le letture supplementari costituiscono uno strumento di apprendimento molto prezioso.

Gli strumenti necessari

Avremo bisogno di un *assembler* (assemblatore), commerciale o di pubblico dominio. Il primo programma assembler offerto dalla Commodore era stato creato dalla Metacomco e funziona ancora bene. La maggior parte dei compilatori C vengono forniti con un assembler che fa parte del pacchetto. Esistono anche svariati assembler 'liberamente distribuibili'. Uno, scritto da Wes Howe, presenta alcuni piccoli bug e un altro, scritto da Charlie Gibbs, risulta completo di tutte le caratteristiche normali e abbastanza esente da bug nella versione più recente (1989).

Entrambi questi due programmi dovrebbero essere disponibili, gratis, presso un club di utenti o in una bbs o su dischi di pubblico dominio (Fish 81 per quello di Howe e Fish 110 per quello di Gibbs. Consultate un catalogo aggiornato dei dischi di Fred Fish per avere le versioni di più recente pubblicazione. NDT). Svariati assembler commerciali sono stati pubblicati dalla Abacus (AssemPro), da Compute! (ASM68010), dalla HiSoft (Devpac, vedi recensione in questo numero) e da altri ancora.

A proposito dei file *include*, questi vengono normalmente forniti come parte del pacchetto dell'assembler. Non preoccupatevi di questi per il momento, ma tenete presente che sarete felici di averne una serie recente e a posto quando vorrete spingervi un poco più a fondo nella programmazione in assembly. Man

mano che progrediremo nel corso prenderemo in esame quelli che ci serviranno.

Avremo bisogno di un *linker*. Questo sarà quasi certamente BLINK, lo splendido programma creato da John Toebes della Software Distillery. I programmi che scriveremo verranno assemblati in file *oggetto* (object); questi ultimi non possono essere eseguiti fino a quando non vengono *linkati* (passati attraverso al linker) e resi *eseguibili* (executable). A un primo esame questo potrebbe sembrare un inutile passaggio in più, ma impareremo ad apprezzarlo in seguito.

Avremo bisogno di un *debugger*. Questo programma è l'equivalente di quello che veniva chiamato *monitor per il linguaggio macchina* sui computer Commodore a 8 bit. Il debugger originale per Amiga della Commodore si chiamava Wack ed è stato presto seguito da altri programmi commerciali, fra i quali spicca MetaScope. Il pacchetto AssemPro della Abacus dispone di un debugger integrato e un eccellente debugger di pubblico dominio, al momento ancora sotto revisione, è stato prodotto da Jim Thibodeau e Larry La Plume.

Avremo bisogno di *manuali di riferimento*. Ai principianti conviene comperare solo un libro: *The AmigaDOS Manual* (Bantam Computer Books). Sebbene il libro sembri, a una prima ispezione, solamente una guida al CLI, scoprirete ben presto che contiene una sezione intitolata *AmigaDos Developer's Manual* che racchiude praticamente tutto quello di cui avrete bisogno per cominciare a esplorare.

Il migliore testo di riferimento per il 68000 viene dal produttore del chip. Il libro 'The MC68000 User's Manual' (MC68000-UM/AD) è pubblicato dalla Motorola. Potreste trovare interessante anche un altro libro della stessa casa, chiamato 'Family Reference Guide' (MC68000-FR68K-D).

Impostazione del corso

Noterete che all'inizio del corso terrò il mio naso molto vicino al codice e che non userò file include o definizioni esterne. Questa impostazione cambierà presto, ma finchè rimane mi aspetto che esaminiate molto attentamente il programma assembly che avete per le mani. Esistono grandi differenze fra i pacchetti assembler disponibili e io non posso scrivere del codice che soddisfi tutti. Dovrete decidere da soli i requisiti che il vostro pacchetto deve avere.

Starò lontano dalle caratteristiche proprie di Amiga per un certo tempo. Cose come la grafica e il suono verranno più tardi. L'enfasi iniziale sarà concentrata sul controllo di flusso di testo. Comincieremo a leggere e a scrivere sullo schermo e sul disco.

Mi sembra che la gestione del testo rappresenti un buon punto di partenza. Essa ci permetterà infatti di chiedere dei dati, mandare messaggi e fare una serie di altre cose utili. Se siete ipnotizzati dalla grafica, dal suono, dal blitter e dal copper, rassegnatevi perché non li vedrete trattati qui per ancora un certo tempo.

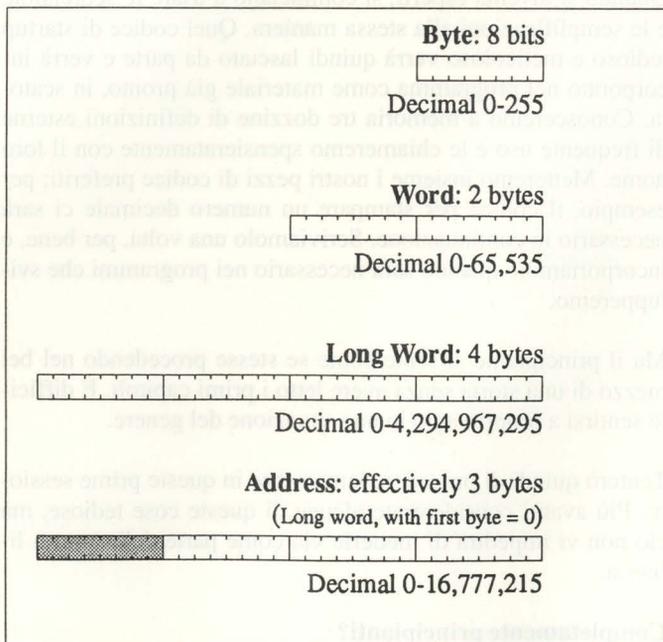
Nozioni di base

La memoria è organizzata in *byte*, come nella maggior parte dei microcomputer che potreste avere incontrato. Il chip 68000 può tranquillamente leggere e memorizzare singoli byte nella sua memoria, utilizzando qualsiasi indirizzo valido. Esiste tuttavia un modo più potente di accedere alla memoria: due byte alla volta.

Due byte adiacenti possono essere letti o memorizzati in una singola mossa, purché il byte inferiore (come indirizzo) si trovi a un indirizzo pari. Questo paio di byte viene chiamato *word* e costituisce un sistema molto efficiente per usare i dati in memoria. Possiamo riferirci a entrambi i byte con una singola istruzione alla quale dovremo fornire l'indirizzo di quello più in basso (inferiore). Questo indirizzo dovrà essere pari.

Per fare un esempio, possiamo leggere (o scrivere) il contenuto degli indirizzi 20 e 21 in una sola azione. Ma non possiamo fare riferimento ai contenuti dei byte 21 e 22 in una sola volta; se il nostro programma tenta di compiere questa operazione, il computer andrà in guru segnalando un *address error* (errore di indirizzamento).

Possiamo anche riferirci a quattro byte alla volta; questo gruppo di quattro byte viene chiamato *long word*. Il computer non può gestire una long word in una sola azione, quindi eseguirà due passaggi per portare a termine l'operazione. Ancora una volta, noi dobbiamo fornire solo l'indirizzo pari del byte inferiore. In questo modo, una operazione su long word all'indirizzo 4 coinvolgerà, in pratica, i quattro byte agli indirizzi 4, 5, 6 e 7.

Lo schema seguente mostra le tre configurazioni di dati, più una quarta: gli *indirizzi* (address). Questi, in genere, occupano una long word. Dal momento che gli indirizzi sul 68000 solo

Tipi di dati del 68000

larghi solo 24 bit (tre byte), la parte in alto della long word non viene utilizzata e dovrebbe essere messa a zero.

I registri del 68000

Dentro ogni microprocessore si trovano zone, riservate al calcolo e all'immagazzinamento, chiamate *registri*. Nel 68000 ce ne sono un certo numero. Faremo un breve accenno a due registri speciali, quindi ci sposteremo sui registri principali, normalmente utilizzati per lavorare.

Il registro PC è il *Program Counter*; esso ci dice dove il computer sta lavorando, istante per istante. Il registro SR è lo *Status Register*, che contiene informazioni (chiamate *flag*) sulle condizioni del computer e dei suoi dati. Ne parleremo ancora, ma per adesso ci spostiamo nell'area principale dove si svolgono le operazioni: i registri dati (Data) e indirizzi (Address).

Ci sono otto *registri dati* (Data Registers), chiamati da D0 a D7. Possiamo utilizzarli in svariati modi: con byte, word o long word. Se usiamo un registro dati per un'operazione su byte, per esempio, verrà modificato o esaminato solo il suo byte inferiore. Un'operazione su word utilizzerà solo i due byte inferiori del registro, mentre una su long word lo userà, come possiamo aspettarci, tutto quanto.

Siamo liberi di usare qualsiasi registro dati per qualsiasi scopo ci piaccia. È utile sapere che il registro D0 è quello più spesso usato per memorizzare i dati che una subroutine, chiamata in precedenza, restituirà. Per questa ragione, faremo meglio a lasciare D0 relativamente libero. In aggiunta, il contenuto dei registri D0 e D1 possono essere alterati dalla chiamata a una subroutine di sistema, così è meglio non tenervi dati che servono per un tempo lungo.

Ci sono otto *registri indirizzi* (Address Registers), chiamati da A0 ad A7. Normalmente questi vengono usati con long word,

D0 A0
D1 A1
D2 A2
D3 A3
D4 A4
D5 A5
D6 A6 Special (library)
D7 A7 Special (stack)

Registri dati e indirizzi del 68000

quindi non si possono usare per operazioni su byte. È possibile usarli per operazioni su word, ma, in quest'ultimo caso, il valore contenuto nella word viene espanso in modo da riempire l'intero registro indirizzi.

Il registro A7 è speciale. Per prima cosa è un registro 'doppio': esiste infatti una seconda copia del registro A7 che non vedremo mai e della quale non ci preoccuperemo fino a quando non arriveremo a trattare roba pesante e potente come il *modo supervisore* (supervisor mode).

In secondo luogo, il registro A7 è il puntatore allo stack principale del nostro programma. Possiamo utilizzarlo, ma trattiamolo con rispetto. Se lo lasciamo stare, saprà badare a sè stesso senza problemi.

Il registro A6 ha un significato speciale su Amiga. Deve essere utilizzato come *puntatore di libreria* (library pointer) tutte le volte che si chiama una libreria Amiga (e questo avverrà spessissimo). Riservate il registro A6 per questo utilizzo; avremo a che farci quanto prima.

Gli altri registri da A0 ad A5 possono essere utilizzati liberamente in qualsiasi maniera ci piaccia. Teniamo a mente che il contenuto dei registri A0 e A1 può essere modificato dalle subroutine di sistema, quindi tenteremo di tenerli abbastanza puliti.

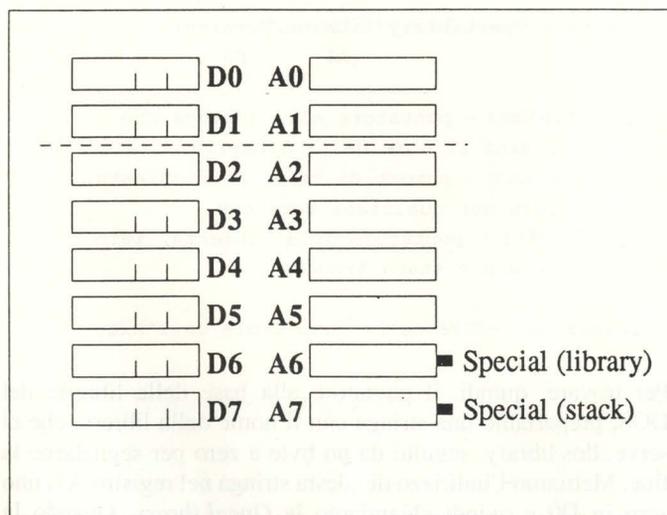
Se avete programmato sui computer Commodore a 8 bit, potrete trovare utile il fatto di pensare ai registri indirizzi in termini del funzionamento della pagina zero sul 6502. Dovremo sicuramente utilizzarli quando avremo a che fare con l'indirizzamento indiretto (ma non per questa volta).

Man mano che cominceremo a esplorare le librerie incorporate in Amiga, scopriremo che le subroutine si fanno passare gli argomenti nei registri dati e indirizzi con numerazione bassa. Tenderemo quindi a mettere i dati e gli indirizzi, che ci servono per un certo tempo, nei registri a numerazione alta, in modo da ridurre i passaggi inutili.

Un primo programma

Caspita, tutta questa teoria e non abbiamo ancora messo le mani su qualcosa di pratico. Prepariamo allora un progetto e occupiamoci di codificarlo. Mi sembra che il programma per principianti più diffuso (negli altri linguaggi) sia quello che stampa sul video 'Hello World!'.

Questo programma girerà solo da CLI, per risparmiarci tutte le complicazioni richieste per un programma che giri anche da Workbench. Lo chiameremo TEST, così quando scriveremo il comando TEST nel CLI o nella Shell, il programma stamperà Hello, World! sullo schermo.

Vedremo che cosa bisogna fare in un approccio che parte dalla base e va verso i livelli superiori, cominciando con il lavoro che dobbiamo fare. Per stampare un messaggio useremo il comando *Write* contenuto nella libreria del DOS.

Il comando Write

La sintassi del comando *Write* è come segue:

```
returnedLength = Write(file,buffer,length)
      D0                D1     D2     D3
D1 - file = file handle
D2 - buffer = puntatore al testo da stampare
D3 - length = intero, lunghezza del testo
D0 - returnedLength = intero, conferma della
lunghezza del testo

Locazione: -$30 nella jump table del DOS
```

Imparerete a leggere questo tipo di sintassi molto velocemente. Dando un'occhiata all'informazione contenuta, vediamo che *buffer* e *length* sembrano abbastanza semplici da capire: quello è il nostro messaggio. Il valore *returnedLength* conferma che il nostro output è stato eseguito con successo.

La *locazione* data è la posizione del comando Write all'interno della tabella di salto (jump table) della libreria. Entreremo nei dettagli di questo in seguito. C'è rimasto solo un pezzetto di informazione che non abbiamo ancora visto: cosa diavolo è un file handle?

Un *file handle* è un identificatore che dice al sistema quale canale di input o di output utilizzare. Questo sta bene, ma come facciamo a procurarci questo numero? Non è così semplice come il numero di periferica che usavamo sul Commodore 64.

Come ci procuriamo il file handle? Se apriamo un file, lo otteniamo automaticamente. Ma adesso abbiamo bisogno di ottenere il file handle del nostro canale di output che esiste già (il CLI dal quale abbiamo eseguito TEST). Nessun problema: usiamo *Output*.

La sintassi di *Output* è:

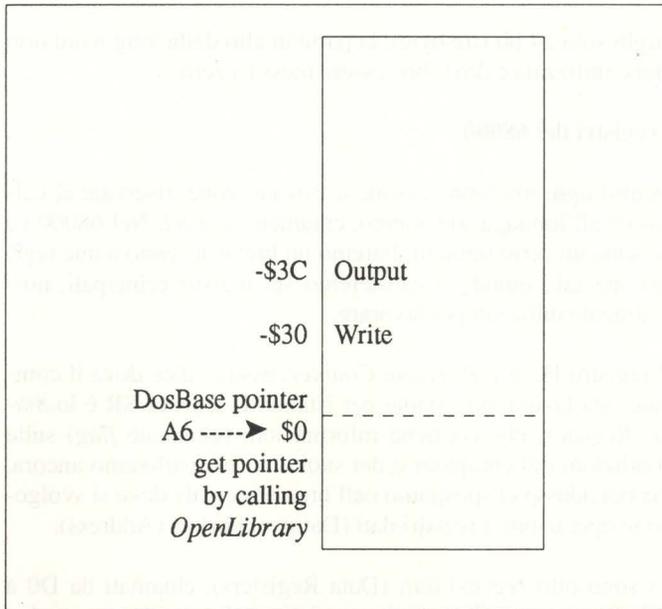
```
    file = Output()
     D0

D0 - file = file handle

Locazione: -$3C nella jump table del DOS
```

Il comando *Write* ci chiedeva di fornire tre parti di informazione. *Output* non ha bisogno di parametri; basta chiamarlo per ricevere il valore del nostro file handle nel registro D0.

Il nostro compito programmatorio adesso sembra più chiaro. Chiamiamo *Output* per avere il file handle del nostro canale di output (lo schermo CLI, a meno che non sia stata usata la redirezione nel lanciare TEST); usiamo quel file handle, più l'indirizzo del nostro messaggio 'Hello World!' e la sua lunghezza, chiamiamo *Write* e via che va.

Ma c'è un piccolo problema. Le due routine di cui abbiamo bisogno, *Output* e *Write*, sono locate in -$3C e -$30 nella jump table della libreria del DOS. Ma dove diavolo è questa libreria?

La jump table della DOS library... da qualche parte in memoria

Librerie e jump table

La libreria del DOS non risiede in alcuna zona fissa della RAM. Probabilmente nel vostro computer sarà in una posizione diversa da quella occupata adesso nel mio. In teoria, potrebbe addirittura non risiedere affatto nella RAM, nel qual caso dovrà esservi caricata. Ancora in teoria, potrebbe addirittura essersi spostata dall'ultima volta che l'abbiamo usata.

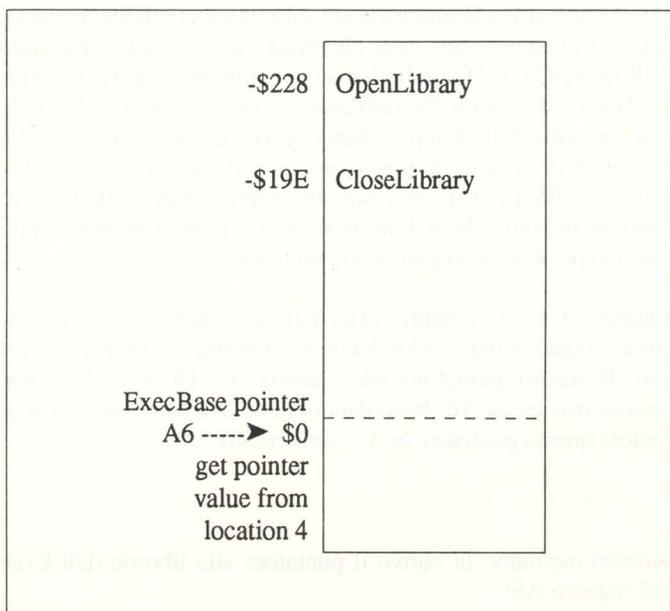
Dobbiamo quindi trovare la libreria, caricarla in memoria se questo risulta necessario e quindi inchiodarla in modo che non si sposti mentre la stiamo usando. Tutte queste operazioni sono svolte da un unico comando: *OpenLibrary*. Più tardi, quando non avremo più bisogno della libreria, dovremo liberarla. Quest'ultima operazione è svolta dal comando *CloseLibrary*.

Questa è la sintassi del comando OpenLibrary:

```
LibPtr = OpenLibrary(LibName,Version)
  D0                   A1      D0

A1 - LibName = puntatore alla stringa che
        indica il nome della libreria
D0 - Version = numero di versione richiesta,
        zero per qualsiasi versione
D0 - LibPtr = puntatore alla libreria, zero
        se non è stata trovata

Locazione: -$228 nella jump table dell'Exec
```

Per trovare, quindi, il puntatore alla base della libreria del DOS, prepariamo una stringa con il nome della libreria che ci serve, dos.library, seguito da un byte a zero per segnalarne la fine. Mettiamo l'indirizzo di questa stringa nel registro A1, uno zero in D0 e quindi chiamiamo la *OpenLibrary*. Quando la funzione restituisce il controllo, il registro D0 contiene uno zero nel caso che la libreria richiesta non sia stata trovata (abbia-

La jump table della Exec library... da qualche parte in memoria

mo probabilmente sbagliato a scriverne il nome), oppure l'indirizzo della libreria del DOS, che è proprio ciò che volevamo.

Potreste avere notato, a questo punto, un apparente paradosso: come facciamo a trovare la libreria dell'Exec? Abbiate fiducia: ci arriveremo senza fatica. Per il momento, seguite il flusso centrale degli eventi.

Una volta trovato il puntatore alla libreria del DOS, possiamo trovare facilmente i punti di ingresso nella jump table per le funzioni *Output* e *Write* utilizzando gli offset da quel punto. Notate che gli offset sono rappresentati da numeri negativi; contiamo dal puntatore in *giù* per raggiungere la casella appropriata nella jump table. Di conseguenza, se il puntatore della libreria del DOS contiene l'indirizzo $C04C70, il salto per la subroutine *Output* verrà effettuato $3C byte più in *basso*, il che corrisponde all'indirizzo $C04C34.

Non preoccupatevi troppo dei numeri; il computer si farà carico dei calcoli aritmetici al nostro posto. A seconda del pacchetto di debug che userete, potrete trovare utile il fatto di sapere che i valori negativi vengono memorizzati dal computer secondo il complemento a due, così che -$3C potrebbe apparire come $FFFFFFC4 quando utilizzate il vostro debugger.

Ricapitoliamo la situazione

Può risultare utile dare un'occhiata al nostro programma così come si è delineato fino a questo punto.

```
1. Trovare la libreria del DOS chiamando
      OpenLibrary
2. Ottenere il file handle chiamando Output
3. Spedire il messaggio chiamando Write
4. Liberare la libreria del DOS chiamando
      CloseLibrary
```

Quest'ultima funzione non è ancora stata ben documentata, quindi eccone la sintassi:

```
CloseLibrary(libPtr)
      A1

A1 - libPtr = puntatore alla libreria che
          vogliamo chiudere

Locazione: -$19E nella libreria dell'Exec
```

L'unico pezzo mancante adesso è come facciamo a trovare la libreria dell'Exec? Risposta: facile, il suo indirizzo è sempre contenuto nella long word che comincia all'indirizzo 4.

Dentro il codice

Prima, una veloce nota sul fatto che dobbiamo mettere due stringhe in memoria. Una è il nome della libreria del DOS seguita da uno zero binario. L'altra è il nostro messaggio. Vediamo in anticipo queste due linee di istruzione; non ne avremo veramente bisogno prima della fine del nostro programma.

```
DosName  dc.b 'dos.library',0
Message  dc.b 'Hello, World!',$0a
```

Le etichette *DosName* e *Message* contrassegnano l'indirizzo al quale iniziano le rispettive stringhe. Il comando dell'assembler DC.B sta per *define constant, byte* (definizione di costante, byte) e significa: adesso viene un gruppo di byte che devono essere messi nella memoria. Questi byte possono essere caratteri come dos.library (fate attenzione alle minuscole e alle maiuscole) o valori come lo zero binario che serve a terminare la prima stringa. La seconda stringa non ha bisogno di terminare con uno zero; ne conteremo i caratteri e diremo a *Write* quanti mandarne.

Non mettete ancora queste linee nel vostro programma. Le abbiamo viste adesso così che possiate sapere cosa sono quando verrà il loro turno.

Invadiamo l'Exec

L'indirizzo della libreria dell'Exec è memorizzato all'indirizzo 4 (long word). Quando usiamo una libreria di Amiga, prima dobbiamo mettere il puntatore alla libreria nel registro A6. Facciamolo.

```
MOVE.L  $4,A6
```

Muovi, long word, il contenuto dell'indirizzo 4 nel registro A6. Adesso il puntatore all'Exec è in A6.

Prima di chiamare *OpenLibrary* dobbiamo mettere il puntatore al nome della libreria da aprire in A1. Una buona strada per mettere un indirizzo in un registro consiste nell'utilizzare il comando LEA (Load Effective Address, carica l'indirizzo effettivo). Potremmo scrivere LEA DosName,A1 e questo funzionerebbe. Ma saremo più specifici e scriveremo:

```
LEA     DosName(PC),A1
```

La notazione PC significa questo: trova l'indirizzo calcolandone l'offset da questo punto del programma. Sarà all'incirca cinquanta byte più avanti, ma il computer si prenderà cura di questi dettagli. Specificando PC risparmieremo un po' di memoria, sebbene il programma funzionerebbe comunque venisse calcolato l'indirizzo della stringa corrispondente all'etichetta *DosName*.

Adesso dobbiamo occuparci della versione della libreria. Ci va bene qualsiasi revisione potremmo trovare, quindi useremo uno zero. Possiamo mettere il valore zero nel registro D0 in molti modi diversi. Per esempio CLR.L D0 (clear long word, pulisci long word) andrebbe bene. Noi useremo il comando MOVEQ (move quick, muovi velocemente).

```
MOVEQ   #0,D0
```

Adesso siamo pronti a chiamare la funzione della libreria. Il registro A6 contiene il puntatore alla libreria. JSR (A6) chiamerebbe una subroutine (Jump Subroutine) che inizia a quell'indirizzo, ma questo sarebbe folle; salteremmo completamente la jump table! Notate l'uso delle parentesi per segnalare un indirizzo indiretto; non saltiamo in A6, ma all'indirizzo contenuto in A6. Ma non vogliamo saltare là; vogliamo invece usare l'offset di -$228 da quell'indirizzo per chiamare la *OpenLibrary*. Ecco quindi cosa scriviamo:

```
JSR     -$228(A6)
```

Questa istruzione chiamerà la funzione *OpenLibrary* contenuta nella libreria dell'Exec. Ma per carità, non dimenticate A6, che punta alla libreria. Omettetelo per distrazione e salterete nell'oblio. Il sistema potrebbe andare in crash così velocemente da non vedere neanche il guru.

Quando la routine *OpenLibrary* ritorna, il registro D0 dovrebbe contenere il puntatore alla libreria del DOS. In caso contrario, avrete probabilmente sbagliato nello scrivere il nome della libreria, o dimenticato lo zero binario alla fine. Controlliamo in ogni caso, mentre spostiamo il puntatore alla libreria in A6:

```
MOVE.L  D0,A6
```

L'indirizzo che avevamo ottenuto si trova adesso in A6. Ma cosa succede se è zero? In questo caso, meglio saltare tutto il resto del programma e uscire. Dopo quest'ultima istruzione, il flag Z (zero) sarà settato (alto, a uno) se tutti i bit spostati erano a zero. Così possiamo controllare con l'istruzione BEQ (branch equal, salta se uguale). Sappiamo che questo salto sarà abbastanza corto, quindi identifichiamolo come *short* (corto):

```
BEQ.S   Exit
```

Breve salto se uguale (a zero) all'etichetta *Exit*, una linea di istruzione che vedremo fra poco. A questo punto mettiamo il resto del nostro lavoro in una subroutine e chiamiamolo con BSR (branch subroutine, salta alla subroutine). Ancora una volta, il salto sarà breve:

```
BSR.S   Main
```

Perchè abbiamo chiamato questa subroutine con BSR (branch), quando abbiamo fatto una chiamata poco prima utilizzando JSR (jump)? È un fatto di vicinanza. Sappiamo che la subroutine *Main* si troverà nelle vicinanze, quindi l'istruzione Branch, più compatta della Jump, ci farà risparmiare della memoria. Le chiamate di sistema non possono essere fatte utilizzando l'istruzione BSR per diverse ragioni, la più semplice delle quali consiste nel fatto che la libreria si troverà probabilmente troppo lontana per essere raggiunta con un branch.

Quando il nostro compito principale è completato, faremo ritorno in questo punto, chiuderemo la libreria del DOS e usciremo. Il nostro puntatore alla libreria del DOS si dovrebbe ancora trovare in A6. Ricordandoci che la *CloseLibrary* vuole vedere questo puntatore in A1, scriveremo:

```
MOVE.L  A6,A1
```

Adesso mettiamo di nuovo il puntatore alla libreria dell'Exec nel registro A6:

```
MOVE.L  $4,A6
```

Siamo pronti a chiamare la *CloseLibrary*, contenuta nell'Exec all'offset -$19E:

```
    JSR     -$19E(A6)
Exit:
    RTS
```

Il guscio esterno è pronto. Adesso possiamo concentrarci sul cuore che dobbiamo creare.

Breve interludio

Il codice che abbiamo appena visto prepara tutte le cose in modo da permetterci di svolgere il nostro compito principale. Per compiti simili potremo utilizzare lo stesso codice introduttivo, per preparare la base di lavoro.

Questo tipo di approccio viene chiamato *startup-code* (codice di inizializzazione). Lo usiamo (o ne usiamo una versione un po' più complessa) sempre, praticamente per tutto quello che facciamo. Dopo un po', tutto questo diventa noioso. A questo punto potremo fare una qualsiasi delle seguenti cose: metterlo in un file e portarlo dentro al nostro programma tramite l'istruzione *include*; metterlo in una macro e quindi farne un pezzo di programma in scatola; oppure assemblarlo e portarlo dentro al programma con il linker, quando ci serve.

Dopo avere scritto qualche dozzina di programmi, non vorrete o non avrete più bisogno di occuparvi del codice di startup nella maggior parte dei casi. Vi tufferete direttamente nel cuore del programma e assumerete che il codice di startup in scatola funzioni correttamente.

Noi, comunque, daremo un'occhiata a questo codice per le prime volte. Di tanto in tanto è bello potere mettere le mani in questa zona e personalizzarla per i propri bisogni.

Il cuore del programma

Il codice di startup chiama questa parte del programma come una subroutine (con un BSR). Il registro A6 contiene il puntatore alla libreria del DOS. Siamo pronti a procedere. Per prima cosa chiamiamo la funzione *Output* per ottenere il nostro file handle. Non c'è bisogno di preparare alcun parametro:

```
Main:
      JSR      -$3C(A6)
```

Il file handle si trova adesso in D0. Ne abbiamo bisogno in D1 per la funzione *Write*:

```
      MOVE.L  D0,D1
```

Adesso dobbiamo mettere l'indirizzo di 'Hello, World!' in D2. Il comando LEA (Load Effective Address) funziona solo con i registri indirizzi, quindi effettueremo l'operazione in due passaggi:

```
      LEA      Message(PC),A0
      MOVE.L   A0,D2
```

Conteremo i caratteri contenuti nella stringa-messaggio manualmente, per questa volta. Includendo il punto esclamativo e il seguente carattere di newline (nuova linea, $0A), abbiamo 14 caratteri, quindi carichiamo velocemente questo numero, scrivendo:

```
      MOVEQ    #14,D3
      JSR      -$30(A6)
```

Abbiamo fatto la nostra chiamata a *Write* e il nostro compito è quindi finito. Ritorniamo al codice di startup, scrivendo:

```
      RTS
```

Infine, ecco le due stringhe che abbiamo menzionato prima:

```
DosName   dc.b  'dos.library',0
Message   dc.b  'Hello, World!',$0a
```

Questo è tutto quello che serve. Rivediamo adesso il programma nel suo insieme, con un po' di istruzioni *equate* per sostituire gli offset delle jump table con qualcosa di più leggibile e con l'aggiunta di un paio di commenti.

La maggior parte degli assembler richiede che le etichette (chiamate a volte anche simboli) siano allineate col margine sinistro. Le linee senza etichetta dovrebbero essere indentate sulla destra. Nel listato che segue, questa convenzione viene rispettata. Le regole che riguardano le linee di commento, che cominciano con un punto e virgola, variano a seconda dell'assembler; io le ho indentate per rendere migliore la leggibilità dell'insieme.

```
      ; programma Hello
      ; codice di startup adatto solo al CLI

      ; equate per la libreria dell'Exec
_LVOOpenLibrary     equ     -$228
_LVOCloseLibrary    equ     -$19E
      ; LVO significa Library Vector Offset

  MOVE.L  $4,A6              ; base dell'Exec
  LEA     DosName(PC),A1     ; puntatore a nome della
                             ; libreria del DOS
  MOVEQ   #0,D0              ; qualsiasi versione
  JSR     _LVOOpenLibrary(A6)
  MOVE.L  D0,A6              ; puntatore a base DOS
  BEQ.S   Exit               ; zero, esci
  BSR.S   Main               ; fai il tuo lavoro
  MOVE.L  A6,A1              ; puntatore a base DOS
  MOVE.L  $4,A6              ; puntatore a base Exec
  JSR     _LVOCloseLibrary(A6)

Exit:
  RTS

Main:
      ; parte principale

      ; equate per la libreria del DOS
_LVOOutput          equ     -$3C
_LVOWrite           equ     -$30
      ;
  JSR     _LVOOutput(A6)     ; prendi output handle
  MOVE.L  D0,D1              ; metti handle in D1
  LEA     Message(PC),A0     ; indirizzo messaggio
  MOVE.L  A0,D2              ; mettilo in D2
  MOVEQ   #14,D3             ; lunghezza messaggio
  JSR     _LVOWrite(A6)      ; spedisci messaggio
  RTS
DosName   dc.b  'dos.library',0
Message   dc.b  'Hello, World!',$0a
  end
```

Mettiamo tutto insieme

Digitate questo listato usando il vostro editor preferito e salvatelo usando il nome TEST.ASM o TEST.S. Fate molta attenzione alle maiuscole e alle minuscole quando scrivete le etichette o i nomi delle librerie e delle funzioni.

Se avete il file *Amiga.lib*, che si trova sul disco degli sviluppatori o negli assemblatori commerciali, potreste cambiare le linee *equ* in qualcosa di più conveniente. Potreste usare un *xref* (external reference, riferimento esterno) al posto di equ e lasciare al linker il compito di trovare il giusto valore di offset. La prossima volta vedremo come fare. Per il momento, il programma stampato qui è completo e non richiede supporto esterno.

Continua a pagina 56

Breakpoint

Quinta parte della serie di articoli sul debugging

di Victor A. Wagner Copyright (c)1989 Victor A. Wagner, Jr.

Vic Wagner iniziò ad avere a che fare con i calcolatori nel 1965 quando divenne parte di un gruppo di studio della US Air Force sulla simulazione digitale del volo. Tornato un civile nel 1966, ha lavorato in quell'epoca, principalmente con costruttori di minicomputer, nel campo del software per sistemi in tempo reale. Nei giorni feriali dalle 8 alle 5, Vic cura la consulenza tecnica telefonica per la Computer Automation Inc. e la manutenzione del software di tre sistemi in tempo reale. Alla sera e nei fine settimana, trascorre il suo tempo conversando con Taarna (il suo Amiga 1000), scrivendo programmi e collegandosi ad AmigaForum. Vic è, inoltre, un'autorità riconosciuta per quanto riguarda il famoso programma di debugging MetaScope prodotto dalla Metadigm Inc.

In quest'ultima puntata daremo un'occhiata agli ultimi due debugger disponibili sul mercato dei prodotti commerciali per Amiga. Questi due programmi sono presenti nell'arena dei debugger Amiga sin dai primi tempi e l'interfaccia utente di uno differisce da quella dell'altro nella massima misura possibile. DB, prodotto dalla Manx Software Systems, usa principalmente un'interfaccia testuale; MetaScope, prodotto dalla Metadigm, è completamente immerso in Intuition (menu, gadget, punta e seleziona). Entrambi sono ampiamente equivalenti per quanto riguarda le prestazioni.

Prima di continuare, dovrei far presente che ho usato MetaScope fin da prima che fosse ufficialmente lanciato sul mercato e che nutro interessi finanziari ed emotivi nel suo successo. Conosco l'autore da più di dieci anni e durante tutto l'arco di tempo occupato dalla progettazione e dall'implementazione di MetaScope, abbiamo passato parecchi intervalli del pranzo ingozzandoci di pizza (l'unico *vero* cibo per un programmatore) e discutendone le caratteristiche e le potenzialità. Ho effettuato prove molto severe e accurate di MetaScope e sono l'autore del testo dimostrativo presente nell'edizione attuale. Non sorprende, quindi, il fatto che MetaScope segua (fino a un certo punto) i miei pregiudizi per quanto riguarda le tecniche di debugging, sebbene alcune sue caratteristiche mi portino a distrarmi. Mi sforzerò di darne un'immagine che sia il meno influenzata possibile.

Okay, diamo un'occhiata all'aspetto esterno dei debugger prima di farci irretire dalle loro caratteristiche interne.

```
MetaScope      81336 ----rwed 17-Jan-89 11:32:06
db             60360 ----rwed 18-Jun-87 17:30:48
```

Le versioni dei miei programmi sono:

```
Aztec C Debugger - Version 3.4b - 6-16-87
MetaScope: The Debugger (Version 1.23)
```

Questa versione di MetaScope è una versione preliminare 'beta' e al momento è compatibile con i sorgenti scritti per il compilatore Lattice. Per il giorno in cui leggerete questo articolo, dovrebbe essere compatibile anche con quelli scritti per il compilatore Manx.

Come possiamo vedere, MetaScope è all'incirca del 35 per cento più grande sul disco. In memoria, i programmi presentano queste dimensioni:

```
MetaScope:                     DB:
indirizzo  Hex    Dec          indirizzo  Hex    Dec

0267D8      D4    212          86EBF8       4      4
87A5D0    3398  13208          87A5D0    2D4C  11596
8CDAC8   103C0  66496          87D328    31B4  12724
         =====  =====          8921B0    32D0  13008
         1382C  79916          895B70     B14   2836
                               89E8D0    22C8   8904
                               8B80C0    24CC   9420
                               8BA598    2028   8232
                                        =====  =====
                                        104A4  66724
```

MetaScope è all'incirca del 20 per cento più grande quando si trova in memoria. Ricordate che Amiga effettua il caricamento in ordine sparso, quindi gli indirizzi hanno poco significato.

L'altra differenza fisica risiede nei manuali. Il manuale di DB è di 42 pagine, provvisto di foratura a tre buchi e fornito insieme al manuale del compilatore nell'unico raccoglitore ad anelli. Il manuale di MetaScope è di 35 pagine sotto forma di libretto a sè stante. Nessuno dei due è fornito di indice, ma entrambi hanno una tavola dei contenuti e sono scritti secondo lo stile che contraddistingue i manuali di programmi per computer: so-

migliano più a dei dizionari piuttosto che a dei racconti.

Considerando le notevoli difficoltà nelle quali mi sono imbattuto nello scrivere di debugging in generale, non posso che immedesimarmi con gli autori di questi due manuali. Ci sono un gran numero di informazioni da impartire al lettore e la maggior parte di queste non sono correlate o omogenee. Tutti i debugger assomigliano a una collezione di cose che i programmatori hanno trovato essere loro utili lungo il corso di molti anni. Quanto detto non va preso come una critica negativa dei due manuali, ma piuttosto come spiegazione del fatto che sono un po' sconnessi.

Una consultazione anche solo saltuaria dei manuali farà emergere chiaramente le grosse differenze che esistono nell'interfaccia utente dei due programmi. Quella di DB è strettamente testuale, come quella di molti debugger che l'hanno preceduto. DB parte con una finestra, un messaggio e il cursore. Il controllo su un programma viene acquistato mediante l'uso del comando *al* o *aL.*· Questo mette in atto alcuni meccanismi di intercettazione nell'AmigaDOS e assume il controllo del prossimo programma lanciato da CLI o da Workbench, stampando il contenuto dei registri e attendendo ulteriori istruzioni.

MetaScope, d'altro canto, apre una cosiddetta *status window* (finestra di stato) e poi attende istruzioni. Non appare nemmeno un posto dove scrivere qualcosa. MetaScope (d'ora in poi abbreviato in MS), comunque, è un programma molto 'Amighizzato'; sulla sua finestra si trovano svariati menu, uno dei quali ci permette di caricare un programma. MetaScope permette inoltre di fornire il nome del programma da debuggare, completo della sua serie di argomenti, sulla stessa linea di invocazione di *MS:*

```
1>MetaScope df0:projs/aargh -a -b -c
```

L'esempio sopra riportato è stato preso da manuale di MS. DB, invece, ha un file di default che viene letto quando il programma viene invocato. Se questo file contiene la singola istruzione *al* (Amiga, carica con simboli) allora le due istruzioni seguenti produrranno il medesimo effetto precedentemente visto per MS:

```
1>db
1>df0:projs/aargh -a -b -c
```

Anziché spiegarvi passo per passo cosa possono fare entrambi i debugger, farò un elenco delle cose che l'uno può fare e l'altro no.

Per prima cosa, dovrei far notare che entrambi sono dotati di notevoli capacità di elaborare espressioni. Il manuale di MS dedica 2 pagine e mezza per dettagliare le proprie (assomigliano molto a quelle del C). Quello di DB vi dedica circa tre pagine. Non scommetterei una lira sul fatto che uno sia più potente dell'altro.

Okay, addentriamoci nei dettagli. Passeremo in rassegna i comandi di DB approssimativamente in ordine alfabetico e faremo dei commenti sulle cose mancanti in MS.

DB ha molti comandi relativi ad Amiga (cominciano tutti per a*)* per i quali non esistono controparti in MS. Molte (magari tutte) le liste dell'Exec possono essere visualizzate: devices, interrupts, libraries, ports, resources e tasks.

Il comando *an* è particolarmente utile, specialmente se stiamo effettuando il debugging di più task. Quando DB viene lanciato, apre una finestra per lo scambio di comunicazioni. Con questo comando viene aperta un'altra di queste finestre e si possono quindi seguire due programmi contemporaneamente. Naturalmente bisogna usare il comando *al* per catturare il prossimo programma che si fa partire.

DB dispone di breakpoint che controllano il valore di locazioni di memoria e che fermano il programma quando queste locazioni di memoria assumono un valore uguale (o NOT) a quello specificato. Il tempo che DB impiega a rendersi conto che il valore di questa locazione di memoria è cambiata dipende dal modo di funzionamento in cui DB si trova in quel momento. Se stiamo procedendo un passo alla volta (single-stepping), il rilevamento è immediato; se no, questi breakpoint associati a cambiamenti in memoria vengono esaminati all'ingresso e all'uscita di ogni funzione. DB può anche effettuare un'operazione di hash o di checksum su una determinata zona di memoria e mantenere un controllo periodico come quello descritto sopra. Nel manuale viene dato un avvertimento esplicito a non impiegare questo tipo di controllo sui primi 256 byte della memoria perché questo rallenterebbe la capacità di eseguire una singola istruzione alla volta. Esiste un comando speciale per tenere sotto controllo i primi 256 byte della memoria.

DB possiede una caratteristica che, sono sicuro, ogni programmatore ha desiderato a un certo punto della sua attività. Possiamo infatti fornire l'indirizzo di una funzione che verrà chiamata *ogni* volta che DB riacquista il controllo. Se la funzione restituisce zero, allora è come se non fosse successo nulla. Ma se il valore restituito è diverso da zero, il programma viene interrotto e viene visualizzato un messaggio contenente il numero in questione.

Con DB possiamo usare due breakpoint temporanei, sia da soli sia insieme ai normali breakpoint contenuti nell'apposita lista.

Facendo eseguire una serie di comandi al raggiungimento di un breakpoint, si possono visualizzare determinate locazioni di memoria quando viene sospesa l'esecuzione del programma.

Nel manuale di DB almeno sei pagine sono interamente riservate al comando *print.* Quest'ultimo ci permette di formattare i dati in tutti i modi possibili e immaginabili. I formati sembrano quasi essere un'estensione di quelli di *printf*() e sono altrettanto criptici.

Ci sono comandi per allocare (forse per i patch), confrontare, riempire e spostare la memoria e si possono salvare fino a 26 macro (una per ogni lettera sulla tastiera). Quest'ultima caratteristica dovrebbe farci risparmiare un bel po' di tempo, in quanto l'operazione di richiamare una macro dalla memoria sarà sempre molto più veloce di dovere digitare l'equivalente da tastiera.

L'input/output può essere redirezionato in modo da leggere i comandi da un file e da mandare l'output in un altro file anziché sul video. Risulta particolarmente utile la capacità di mandare solo i *comandi* in un file, in modo da poterli rileggere in un secondo tempo e ricostruire le operazioni che si stanno facendo ora.

La prima cosa che si nota, cominciando a giocherellare con MS, è che non sembra esserci un limite al numero di finestre di output che si possono aprire. Sebbene questo sia fondamentalmente vero (dal momento che ogni finestra occupa un *poco* di memoria, esiste comunque un limite stabilito dalla quantità di memoria disponibile), l'utilità di avere più finestre di tipo *break*, *hunk*, *status* o *symbol* è limitata.

Avere più finestre di tipo *memory*, invece, cambia il proprio modo di vedere il debugging. Se eravate abituati a dare qualche tipo di comando per visualizzare della memoria ogni volta che veniva raggiunto un breakpoint, questo diventa completamente inutile in MS (oltre che impossibile). Basta semplicemente aprire un finestra sulla memoria da controllare e MS la visualizzerà. Prima che possiate dire "Aspetta un momento, io uso i miei comandi per visualizzare qualcosa basandomi sul contenuto di un registro o di una variabile", devo segnalare che MS permette di usare al massimo la sua capacità di elaborare espressioni, proprio per determinare l'indirizzo di partenza di quella finestra. L'espressione può essere sia *statica* (fissata al momento del suo inserimento), sia *dinamica* (determinata ogni volta che il contenuto della finestra deve essere visualizzato). Espressioni che risultano particolarmente utili sono: **pc** - il program counter attuale, utile per vedere dove si trova in questo momento il programma e **sp** - lo stack pointer attuale, per potere esaminare tutti gli argomenti di una funzione e altro ancora.

Naturalmente, vorremo che queste espressioni siano 'dinamiche' in modo da poter vedere la memoria man mano che i registri cambiano. L'uso della parola 'dinamico' potrebbe indurre in un piccolo errore, dal momento che le espressioni che contengono registri vengono calcolate solo quando MS ha il controllo del programma. Sebbene questo ci permetta lo stesso di guardare la memoria a cui si riferiscono i registri, non spreca tanto tempo a ricostruire il display delle varie zone di memoria mentre il nostro programma sta girando.

Vi sento chiedere 'Ma allora sicuramente MS non visualizza i cambiamenti che avvengono in memoria mentre il mio programma sta girando e mentre MS non ha il controllo delle cose'. Ebbene, questa potrebbe essere la maggiore differenza fra MS e i debugger che abbiamo visto finora.

MS aggiorna l'output delle sue finestre sulla memoria circa dieci volte al secondo, indipendentemente dal fatto che il programma stia girando o meno. Questo significa che possiamo vedere le variabili delle coordinate x e y del mouse, in una struttura Window, cambiare, mentre noi muoviamo il mouse in giro.

Tutte le finestre di MS sono dotate di barre per lo scrolling e di gadget di dimensionamento in modo da poter cambiare la quantità di dati visualizzati o scrollare quà e là. Possiamo inoltre selezionare indirizzi da esaminare o da usare nei breakpoint semplicemente puntandoli e selezionandoli con il mouse.

Sui breakpoint permanenti possiamo anche mettere delle *condizioni*. Queste sono espressioni che vengono valutate ogni volta che il breakpoint associato viene raggiunto. Se l'espressione risulta vera (diversa da zero), allora il contatore associato al breakpoint viene incrementato e, se l'appropriato valore del contatore è stato raggiunto, il programma viene sospeso. Questo significa che MS può produrre breakpoint come: 'fermati la terza volta che passi di qui e che trovi il contenuto del registro D0 uguale all'inizio del buffer di input'.

MS è dotato di un comando *freeze* (congelare) che si può applicare a entrambe le finestre di status e di memoria. Questo mantiene il contenuto di tali finestre costante per una comparazione che si vuole fare più avanti con il contenuto delle stesse finestre in un altro punto del programma. Questo comando può, inoltre, risistemare a piacimento la macchina nelle stesse condizioni contenute nelle finestre congelate. Ciò significa che possiamo salvare le condizioni della macchina (congelando le finestre appropriate) e poi ripristinarle più tardi quando vogliamo tornare indietro e provare ancora.

Possiamo fare registrare automaticamente a MS tutte le sue mosse in un file (su disco o su stampante). Possiamo anche mandare l'intero contenuto di una finestra in questo file.

Con MS possiamo dare i nomi che vogliamo a tutte le nostre numerose finestre, per esempio 'Dove siamo adesso' (per una finestra con indirizzo di base dinamico basato sul contenuto di *pc*) oppure 'Stack' (per una finestra basata dinamicamente sul contenuto di *sp*). Questo può tirarci fuori dalle situazioni più complicate. Naturalmente quando MS scrive il contenuto di una di queste finestre nel file-registrazione, include anche il suo nome.

MS ci permette di cambiare il contenuto di una locazione di memoria o di un registro semplicemente selezionandoli col mouse. A seguito della selezione, appare un requester nel quale possiamo scrivere il nuovo valore. Possiamo inoltre cambiare un'istruzione direttamente in memoria, inserendola in formato assembly. Se abbiamo una finestra sulla memoria possiamo scegliere fra la visualizzazione in formato dati e in formato codice. Nell'ultimo caso possiamo alterare un'istruzione selezionandola con il mouse.

Le prossime due caratteristiche sono presenti solo nella versione più recente (e non ancora pubblicata) di MetaScope. L'autore mi ha assicurato che l'update dovrebbe essere disponibile per il tempo in cui leggerete questo articolo.

MS può salvare una configurazione fatta di: breakpoint permanenti, locazioni delle finestre, dimensioni, espressioni usate per calcolare indirizzi base e status (congelati o meno). In questo modo possiamo preparare la nostra sessione di debugging come più ci piace e poi salvarla, per non doverla reimpostare tutta la prossima volta che iniziamo una nuova sessione.

Continua a pagina 66

Estendiamo l'uso dei nostri comandi

Wildcard in offerta speciale

di Dan Schein

Tutti noi abbiamo dei comandi che ci piacerebbe usare con più argomenti o con le wildcard. Un esempio è il programma MORE di Carolyn Scheppner, fornito con la versione 1.2 e 1.3 del disco Workbench.

Questo programma è un lettore di testo di alta qualità che presenta caratteristiche di funzionamento avanzate, come lo scrolling bidirezionale del testo, la ricerca bidirezionale di stringhe nel testo e (nella nuova versione dell'1.3) la possibilità di modificare il file che si sta leggendo. Più di una volta avrei voluto digitare:

```
MORE pippo.bar pippo.c
```

ma solo il primo file verrebbe visualizzato da MORE. Il secondo argomento rimarrebbe semplicemente ignorato.

Qualche tempo fa, mi sono detto che ci deve essere un modo semplice (e facile) per fare sì che programmi come questo accettino più di un argomento alla volta. Così, dopo un po' di pensamenti, un po' di hacking e qualche cheeseburger sono arrivato a questa soluzione.

Ho chiamato questo programma EXT, in quanto può essere usato come EXTension (estensione) di praticamente qualsiasi altro programma.

Il principio che anima EXT è abbastanza semplice e si capisce meglio se prima si dà un'occhiata alla sua sintassi d'uso:

```
EXT [-e] COMMAND (filename1,...,filenameN).ext

   EXT [-e] COMMAND filename.(ext1,...,extN)

   COMMAND - comando che deve essere eseguito
   filename - nome del file SENZA estensione
   ext - estensione del file
   -e - solo eco, non eseguire
```

Vediamo adesso alcuni esempi:

```
1  EXT MORE df0:msdos/read(me.fnf,.me)

2  EXT MORE src:c/Pnews-(e,m).c

3  EXT ADDBUFFERS DF(0,1): 200
```

Nell'esempio numero 1, EXT viene usato per estendere la stringa 'MOREdf0:msdos/read' con le stringhe racchiuse all'interno della parentesi rotonda, una alla volta, naturalmente. Questo esempio sarebbe uguale a:

```
MORE df0:msdos/readme.fnf

MORE df0:msdos/read.me
```

Nel secondo esempio, EXT viene usato per estendere la parte centrale di una stringa, anziché la parte iniziale o finale. Questo significa che corrisponde ai comandi:

```
MORE src:c/Pnews-e.c

MORE src:c/Pnews-m.c
```

Il terzo esempio mostra la capacità di EXT di passare più di un argomento a un comando. Il numero di argomenti che potete utilizzare è limitato unicamente dalla lunghezza della linea di comando del DOS (DOS_COMMAND_LINE). Per definizione questa stringa è lunga 255 caratteri. Il terzo esempio equivale ai seguenti comandi:

```
ADDBUFFERS DF0: 200

ADDBUFFERS DF1: 200
```

Il numero di argomenti che EXT può gestire è limitato solo dalla lunghezza della linea di comando del sistema. EXT può gestire fino a 254 caratteri su Amiga. L'unica reale limitazione di EXT è che non dovrebbe essere usato con comandi o programmi che necessitano di interagire con l'utente tramite il CLI dal quale sono stati lanciati.

Se, per esempio, digitiamo:

```
EXT diskcopy df0: to df(1,2):
```

il programma Diskcopy parte immediatamente, senza aspettare che l'utente prema il tasto RETURN come normalmente richiesto. Questo è causato dal fatto che il programma che viene esteso non possiede un canale *stdin*. In questo caso, Diskcopy si limita a procedere, ma altri programmi potrebbero comportarsi in maniera differente. Basta tenere a mente che questa limitazione è la stessa di cui soffre il comando RUN fornito dalla Commodore. Se, quindi, un programma o un comando non può essere lanciato con RUN, allora non può essere lanciato neanche con EXT.

Il codice sorgente di EXT

Il programma è stato scritto in modo da poter essere portato facilmente su molti sistemi operativi diversi (vengono in mente MS-DOS e UNIX). Ho messo tutte le cose che potrebbero richiedere cambiamenti in una semplice istruzione *define* all'inizio del listato.

Le uniche altre due parti che potrebbero dover essere cambiate o rimosse sono le linee che precedono e che seguono la nota di copyright (nella routine *print_usage*). Queste due linee cambiano il colore del cursore e potrebbero avere effetti indesiderati su sistemi diversi da Amiga.

EXT è stato scritto e compilato con il Lattice C V5.0. Usando la sintassi contenuta nei commenti di intestazione, EXT può essere compilato in modo da diventare *pure* e da poter quindi essere reso residente con il comando *resident* fornito con il Workbench V1.3.

Il codice sorgente è riccamente commentato in modo che il programmatore C principiante o esperto possa capire la teoria e il funzionamento di EXT. EXT costituisce anche un buon esempio dell'uso dei puntatori C e dei benefici che essi possono apportare a programmi scritti come si deve.

Vorrei ringraziare Carolyn Scheppner per avere scritto MORE, per avere sopportato le mie stupide domande quando imparavo il C e anche per avere ricontrollato questo articolo e il codice sorgente di EXT. I suoi commenti sono stati (come sempre) portatori di riflessione e di grande aiuto.

Copyright

E adesso, per i malintenzionati, le note di copyright. Il codice sorgente di EXT e il risultante eseguibile possono essere usati (e modificati) come desiderate, a patto che non vengano venduti e che non siano inclusi o utilizzati in alcun prodotto commerciale.

Il sorgente di EXT e il suo eseguibile possono essere riversati su BBS e su servizi commerciali e possono essere distribuiti anche su dischi PD/ ShareWare.

Questo è tutto! Se non avete un compilatore C o non volete digitare il codice sorgente, l'eseguibile e il sorgente stesso sono a vostra disposizione per potere essere scaricati dalla *BERKS* AMIGA BBS (USA: 215/678-7691 a 1200/2400 baud). (sono anche disponibili sul dischetto di Transactor relativo a questo numero - ED)

```
/****************************************************

 (C) Copyright 1989 by Sneakers Computing

                    2455 McKinley Ave.
                    West Lawn, PA 19609
                    (215) 678-8984

                    UUCP: sneakers@heimat

 All Rights Reserved

 Questo sorgente e il risultante eseguibile possono
 essere usati (e modificati) per qualsiasi scopo, a
 patto che (#1) non siano venduti e (#2) non siano
 inseriti o usati in un prodotto commerciale. Per
 ulteriori informazioni contattate l'indirizzo
 sopracitato.

 Nome programma: EXT.c

 Scopo:     Questo programma offre un'estensione per
            altri programmi e comandi per mezzo di
            un metodo di sostituzione di stringhe.

 Versione:         Versione 1.00

 Creata:           01/29/1989

 Programmatore:    Dan "Sneakers" Schein

 OS:               AmigaDOS V1.3

 Compilatore:      Lattice V5.0

 Comandi comp.:    Lc -tr -L ext

 Nota: Il flag "-tr" farà si che venga fatto il link
       con "cres.o", rendendo EXT "pure" in modo che
       possa essere usato come comando residente.

****************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

          /*
           * DOS_COMMAND_LINE è il numero massimo di
           * caratteri che il sistema operativo accetta
           * in una linea di comando.
           *
           * L'idea che sta dietro questo programma è di
           * renderlo il più facilmente portabile
           * possibile. Solo i seguenti "ifdef"
           * dovrebbero eventualmente essere modificati
```

```
      * per portare il codice su MS-DOS o UNIX!
      *
      */

#define AMIGADOS

#ifdef  AMIGADOS
#define LEN 3
#define DOS_COMMAND_LINE 255
#endif

     /*
      * Queste definizioni rendono semplicemente
      * il codice più leggibile
      *
      */
#define LPAREN '('
#define RPAREN ')'
#define SPACE  ' '

#ifndef FALSE
#define FALSE 0
#endif

#ifndef TRUE
#define TRUE !FALSE
#endif

void main();
void print_usage();


void main (argc, argv)
int  argc;
char **argv;
{
   int  i, echo_only=FALSE;
   char *lparen, *rparen, *first_arg, *last_arg;
   char *nextdelim;
   char oldcmdline[DOS_COMMAND_LINE];
   char newcmdline[DOS_COMMAND_LINE];

   /*
    * Controlla se l'utente vuole aiuto.
    *
    */
   if ((argc<2) || (argv[1][0] == '?'))
      print_usage();

   /*
    * Guarda se l'utente vuole solo l'eco. Se sì,
    * setta l'apposito flag e incrementa argv in
    * modo da saltare "-e", poi diminuisci argc di
    * uno così corrisponde al restante numero di
    * argomenti argv.
    *
    */
   if ((argv[1][0] == '-') &&
       toupper(argv[1][1]) == 'È) {
      echo_only=TRUE;
      argv++;
      argc--;
   }

   /*
    * Costruisci una stringa con gli argomenti
    * della linea di comando
    *
    */

   for (i=0, oldcmdline[0]=NULL; i<argc; i++)
   {
      strcat(oldcmdline, argv[i]);
      strcat(oldcmdline, " ");
   }

   /*
    * Crea un puntatore alla parentesi tonda
    *
    * NULL se non esiste
    *
    */
   lparen = strchr(oldcmdline, LPAREN);
   rparen = strchr(oldcmdline, RPAREN);

   /*
    * Controlla che la sintassi sia valida.
    * In caso negativo, mostra l'help
    *
    * Gli errori di sintassi sono...
    *
    * Parentesi chiusa prima di parentesi aperta, o
    * non esiste parentesi aperta, o
    * non esiste parentesi chiusa
    *
    */

   if ((rparen<lparen) || !lparen || !rparen)
      print_usage();

   /*
    * Crea un puntatore all'estensione (se esiste)
    *
    */

   last_arg = (rparen+1);
   /*
    * Copia la linea di comando fino (ma non
inclusa)
    * alla parentesi aperta in una nuova stringa.
    *
    * Metti un NULL in fondo alla nuova stringa.
    *
    * Crea un puntatore al primo argomento dopo la
    * parentesi aperta.
    *
    * Incrementa il puntatore alla parentesi aperta.
    *
    */
```

```
   strncpy(newcmdline,oldcmdline,lparen-oldcmdline);
   newcmdline[lparen-oldcmdline]=NULL;
   first_arg=&newcmdline[strlen(newcmdline)];
   lparen++;

   /*
    * Entra in un ciclo (loop), trovando la prossima
    * stringa di sostituzione e quindi aggiungendola
    * alla nuova linea di comando e facendola
    * seguire dall'eventuale testo di estensione.
    *
    * Naturalmente tutto questo è fatto per mezzo
di
    * puntatori, anziché di stringhe vere e proprie.
    *
    * Continua a rimanere nel loop fino a quando
non
    * esauriamo le stringhe di sostituzione,
uscendo
    * poi con il comando BREAK.
    *
    */
   for(;;) {
      *first_arg=NULL;
      nextdelim=strpbrk(lparen, ",)");
      *nextdelim=NULL;
      strcat(first_arg, lparen);
      strcat(first_arg, last_arg);

      if (echo_only)
         printf("ext: %s\n", newcmdline+LEN);
      else if(system(newcmdline+LEN))
         printf("ext: ERROR EXECUTING \"%s\"\n",
                                 newcmdline+LEN);
      lparen=nextdelim+1;
      while (*lparen == SPACE)
         lparen++;
      if (lparen > rparen)
         break;
   }
   exit(0);
}

void print_usage()
{
   char esc = '\33';

   /*
    * Le sequenze di escape utilizzate qui sotto
    * potrebbero causare dei problemi su sistemi
    * operativi diversi da Amiga. Nel caso,
    * rimuovetele.
    */

   printf("%c[0;1;33;40m",esc);
   printf("EXT V1.0 (C) 1989 by Sneakers
Computing\n\n");
   printf("%c[0;31;40m",esc);
   printf("Uso:  ext [-e] COMMAND
(filename1,...,filenameN).ext\n");
   printf("      ext [-e] COMMAND
filename.(ext1,...,extN)\n\n");
   printf("Dove  COMMAND  = comando da eseguire\n");
   printf("      filename = nome del file SENZA
estensione\n");
   printf("      ext      = estensione\n");
   printf("      -e       = solo eco, non
eseguire\n\n");
   printf("Esempi:  ext TYPE (Mail,News).c\n");
   printf("         ext LIST
df0:src/day.(c,bak)\n");
   printf("         ext ADDBUFFERS ff(s,s1,s2):
200\n");

   exit(1);
}
```

Sul prossimo numero:

- Programmare in ARexx
- L'arte della programmazione in linguaggio Assembly
- I dischi di Fred Fish
- Algoritmo veloce di riempimento aree
- L'uso dei gedget multipli
- Lo standard IFF
- Il linguaggio Assembly su Amiga, seconda parte
- Come disegnare usando la libreria grafica da C e da Basic
- Il MIDI
- Notizie dal mondo Amiga
- ViewPort

I Servizi
di

Amiga Transactor offre una serie di servizi per agevolare i propri lettori nel reperimento di software e materiale utile alla programmazione.
È disponibile l'intera libreria di dischetti di pubblico dominio curata da Fred Fish. Ogni dischetto contiene numerosi programmi e utility, spesso corredati da listati sorgenti e commenti degli autori.
Per districarsi fra le centinaia di programmi disponibili nei dischi di Fred Fish, è stato creato un apposito catalogo di 20 pagine. Tale elenco riporta, divisi per categoria e in ordine alfabetico, tutti i programmi presenti, completandoli con informazioni quali la descrizione della funzione, l'autore, il numero di versione, la disponibilità del sorgente e il disco nel quale sono contenuti. I dischetti possono essere ordinati contrassegnando i numeri desiderati, purché la quantità sia un multiplo di cinque. Per ordini amministrativi saremo costretti a non accettare ordini che non soddisfino questa regola. Tutto il materiale, a esclusione dei listati pubblicati sulla rivista, viene fornito nella versione originale americana, senza traduzione o modifiche.
A ogni numero della rivista corrisponde un disco chiamato "AmiTrans Disk" che contiene tutti i listati pubblicati su quel numero, nonché i corrispondenti eseguibili e tutti gli altri programmi di pubblico dominio menzionati negli articoli.

BUONO D'ORDINE

Completa il buono d'ordine (o una sua fotocopia) e spedire in busta chiusa a: I servizi di Transactor per Amiga - Via Rosellini, 12 - 20124 Milano

Si può allegare: assegno, contanti o fotocopia della ricevuta di versamento c/c n. 11666203 intestato a Gruppo Editoriale Jackson.
Non si effettuano spedizioni contrassegno

Desidero ricevere i seguenti articoli; contrassegnare con una X i numeri di Fish disk desiderati (a gruppi di 5)

Nota: Il disco 164 non è disponibile

☐ 1	☐ 12	☐ 23	☐ 34	☐ 45	☐ 56	☐ 67	☐ 78	☐ 89	☐ 100	☐ 111	☐ 122	☐ 133	☐ 144	☐ 155	☐ 167
☐ 2	☐ 13	☐ 24	☐ 35	☐ 46	☐ 57	☐ 68	☐ 79	☐ 90	☐ 101	☐ 112	☐ 123	☐ 134	☐ 145	☐ 156	☐ 168
☐ 3	☐ 14	☐ 25	☐ 36	☐ 47	☐ 58	☐ 69	☐ 80	☐ 91	☐ 102	☐ 113	☐ 124	☐ 135	☐ 146	☐ 157	☐ 169
☐ 4	☐ 15	☐ 26	☐ 37	☐ 48	☐ 59	☐ 70	☐ 81	☐ 92	☐ 103	☐ 114	☐ 125	☐ 136	☐ 147	☐ 158	☐ 170
☐ 5	☐ 16	☐ 27	☐ 38	☐ 49	☐ 60	☐ 71	☐ 82	☐ 93	☐ 104	☐ 115	☐ 126	☐ 137	☐ 148	☐ 159	☐ 171
☐ 6	☐ 17	☐ 28	☐ 39	☐ 50	☐ 61	☐ 72	☐ 83	☐ 94	☐ 105	☐ 116	☐ 127	☐ 138	☐ 149	☐ 160	☐ 172
☐ 7	☐ 18	☐ 29	☐ 40	☐ 51	☐ 62	☐ 73	☐ 84	☐ 95	☐ 106	☐ 117	☐ 128	☐ 139	☐ 150	☐ 161	
☐ 8	☐ 19	☐ 30	☐ 41	☐ 52	☐ 63	☐ 74	☐ 85	☐ 96	☐ 107	☐ 118	☐ 129	☐ 140	☐ 151	☐ 162	
☐ 9	☐ 20	☐ 31	☐ 42	☐ 53	☐ 64	☐ 75	☐ 86	☐ 97	☐ 108	☐ 119	☐ 130	☐ 141	☐ 152	☐ 163	
☐ 10	☐ 21	☐ 32	☐ 43	☐ 54	☐ 65	☐ 76	☐ 87	☐ 98	☐ 109	☐ 120	☐ 131	☐ 142	☐ 153	☐ 165	
☐ 11	☐ 22	☐ 33	☐ 44	☐ 55	☐ 66	☐ 77	☐ 88	☐ 99	☐ 110	☐ 121	☐ 132	☐ 143	☐ 154	☐ 166	

DESCRIZIONE

☐ Fish Disk	Lit. 35.000 per gruppo di cinque
☐ Catalogo	Lit. 15.000 aggiornato fino al Fish 138
☐ AmiTrans Disk #1	Lit 15.000
☐ #2 ☐ #3 ☐ #4	Lit 15.000

Cognome ....

Nome ....

Via ....

Cap. .... Città ....

Prov. .... Tel. ....

Firma ....

(se minorenne quella di un genitore)
Gli ordini non firmati non verranno evasi.

Tutti i prezzi sono da intendersi IVA inclusa e spese di spedizione comprese.

Obbedire alle regole!

La coerenza è la base dell'interfaccia utente

di Andy Dawson

Andy Dowson è un importante dirigente nel campo dei mass-media che ha lavorato con l'Amiga fin dalla sua nascita. È laureato con lode in psicologia e si è specializzato in psicologia dell'interfaccia utente. Andy ha lavorato come consulente in varie organizzazioni di grandi dimensioni, dando consigli sui vari aspetti della presentazione dei computer e dell'interazione con l'utente.

All'inizio la Commodore stabilì alcune regole fondamentali per gli sviluppatori.

Dissero innanzitutto che vi era un solo indirizzo assoluto in tutta la macchina e che si trovava a $4; a qualsiasi cosa si dovesse accedere, bisognava farlo attraverso tale indirizzo, sia che si trattasse di una routine del Kernel situata in ROM, sia di un'area dati di sistema o di una parte dell'hardware. Coloro che osservarono questa regola notarono che il loro software si adattava a diversi modelli e varianti del sistema operativo senza difficoltà. Chi invece ignorò questa norma scoprì che il suo software si disgregava non appena veniva fornita una nuova versione del sistema operativo o veniva introdotta una nuova macchina.

Con Intuition la Commodore creò anche un ambiente base per finestre, icone, mouse e puntatori (sappiamo tutti che si tratta dell'ambiente WIMP). Questa è la base del Workbench, e la maggior parte delle case produttrici di software ha accettato questa regola come un modo gradevole di affrontare i programmi. Ma Intuition non era solo una caratteristica del Workbench, si trattava piuttosto di una completa filosofia Amiga mirata ad enfatizzare e migliorare il concetto di software integrato. Intuition venne progettato per rendere la vita più facile e semplice all'utente finale, e per fare in modo che il sistema Amiga, che include la gamma del software commerciale, fosse un sistema veramente coerente.

I concetti base di Intuition sono le finestre, le icone ed i menu. Si basava sull'idea che l'utente finale non avrebbe dovuto aver bisogno di imparare complesse combinazioni di tasti per avviare le sue applicazioni; così si scelse il sistema "point-and-shoot", ovvero "punta e spara", prima usato dalla Xerox e poi reso popolare dall'Apple Macintosh. L'Amiga Commodore adottò ed estese questo sistema per andare incontro alle necessità di un sistema multi-tasking. La loro intenzione era quella di fornire un metodo intuitivo per tutto il software Amiga, rendendo così il primo stadio della curva di apprendimento meno traumatico di quanto avrebbe potuto essere.

Per l'utente novello tale metodo intuitivo è di immenso valore, e in questo modo molte persone "anti computer" sono state convinte molto velocemente dalla facilità con la quale, per esempio, riuscivano a disegnare con DeluxPaint o a creare semplici documenti con TextCraft. Anche l'utente occasionale di software ne può beneficiare, non ha più bisogno di studiarsi il software sui manuali mentre può creare, basandosi sull'esperienza passata e guidato dall'interfaccia intuitiva.

Tuttavia, se l'utente si limita a un solo programma applicativo o se si tratta di un utente molto regolare, il meccanismo del metodo intuitivo può diventare noioso. Subito si appresterà a costruire una mappa cognitiva della struttura del programma. Automaticamente vengono apprese tutte le vie seguite dal programma (presumendo che sia scritto bene e che sia coerente!) e il mouse potrebbe diventare un ostacolo.

Quasi tutti gli utenti utilizzeranno regolarmente uno o due programmi software, e questi faranno parte di quel pacchetto che è andato oltre lo stadio dell'interfaccia intuitiva. I tasti funzione, le combinazioni con il tasto Amiga e le "Meta-key" diverranno la norma. A dire il vero le caratteristiche più estese verranno ancora richiamate usando i menu o le icone, ma per la maggior parte delle operazioni si potrebbe non utilitzzare il mouse e buttarlo dalla finestra.

Ogni tanto tutti questi utenti spazieranno brevemente verso altri programmi, perché per esempio vogliono pasticciare con un database, o perché improvvisamente scoprono che hanno bisogno di fare un disegno per illustrare il loro testo, o magari hanno bisogno di lavorare con il foglio elettronico che avevano messo da parte sin da quando lo hanno comperato. In questo caso il metodo intuitivo non è secondo a nessuno e, con un'interfaccia utente coerente e costante, l'Amiga e il suo software diventano un pacchetto d'eccezione, inspirando molta fiducia da parte dell'utente.

Questa è la filosofia Amiga.

I Menu di Intuition

Con Intuition si creò una grande quantità di menu e per assicurare la coerenza in tutta la gamma del software Amiga la Commodore stabilì inizialmente alcune regole fondamentali per i due menu che venivano usati più frequentemente. Crearono un menu standard *project* ed un menu standard *edit*.

Il menu project era costituito da:

NEW - il quale crea semplicemente un nuovo progetto

OPEN - il quale riporta al progetto precedentemente salvato

SAVE - salva il progetto attuale sul disco

SAVE AS - salva il progetto attuale con un nuovo nome

PRINT - stampa l'intero progetto

PRINT AS - stampa il progetto ma permette all'utente di alterare i default

QUIT - esce dal programma ma avvisa l'utente se il progetto è stato modificato ma non salvato.

Molte software house hanno accettato questa convenzione, sebbene sembri che alcuni dei pacchetti più grossi l'abbiano ignorata. Per esempio, WordPerfect usa un'insensata ed inutile collezione di sotto menu *retrieve* e *save*, mentre SuperBase ha modificato il menu standard *project* per fare fronte alla natura specialistica del sistema del database.

Altri pacchetti tendono a conformarsi, sebbene gli utenti alle prime armi si confondano spesso quando si trovano di fronte a OPEN di certi pacchetti e LOAD di altri. Una differenza nei nomi implica una differenza nelle operazioni, ed ecco che si distrugge immediatamente il fondamento base del metodo intuitivo.

Il menu edit è molto più appropriato per un word processor o per un programma di disegno, sebbene anche i fogli elettronici, i database, i pacchetti di grafica e musica sarebbero in grado di usarlo.

Lo standard Commodore era:

UNDO - cancella l'operazione precedente (se non è possibile, tale opzione andrebbe disabilitata!)

CUT - rimuove la porzione selezionata e la registra nella ClipBoard

COPY - copia la porzione selezionata nella ClipBoard

PASTE - inserisce nel progetto la copia che si trova nella ClipBoard

ERASE - rimuove la porzione selezionata senza registrarla nella ClipBoard

Ogni progetto che dichiara di essere in grado di lavorare in multi-tasking deve dare un completo supporto alla ClipBoard. Sfortunatamente non sempre è così e quindi i prodotti di molte software house non diventano parte di un sistema integrato semplicemente perché ignorano il concetto di ClipBoard.

La Commodore-Amiga ha chiaramente interpretato la curva di apprendimento intuitivo, e ha anche capito che, con un po' di esperienza, molti utenti sarebbero rimasti attaccati alla tastiera per la maggior parte del tempo. Hanno quindi dato a Intuition la capacità di richiamare le funzioni dei menu utilizzando delle abbreviazioni sulla tastiera. La Commodore non ha così solo stabilito alcune regole base per i menu, ma ha anche dato degli standard per le abbreviazioni su tastiera. Sotto molti aspetti è ancora più importante che si adotti questo standard, poiché le opzioni dei menu, nella maggior parte dei casi, si spiegano da sole e l'utente non ha bisogno di memorizzarle. Tuttavia la cosa principale, per quanto riguarda le abbreviazioni da tastiera, è che queste devono essere memorizzate, quindi la coerenza nei diversi pacchetti è di immensa importanza.

La Commodore ha dato due raccomandazioni per le abbreviazioni da tastiera. La prima riguardava le opzioni dei menu, da usare con il tasto Amiga destro, e questo in effetti era una sostituzione per le funzioni alle quali si accedeva con il tasto destro del mouse. La seconda riguardava una selezione più generale all'interno di un progetto (che normalmente corrisponde alle funzioni del tasto sinistro del mouse) da usare con il tasto Amiga sinistro. Il fatto che gli utenti sarebbero stati in grado di tenere premuto il tasto Amiga con il mignolo di una mano, e premere uno dei tasti normali che di solito premevano con l'altra, faceva parte delle direttive Amiga. In questo modo i dattilografi esperti avrebbero potuto usare le abbreviazioni senza sentirsi impacciati o goffi.

Per quanto riguarda le abbreviazioni dei menu la Commodore raccomanda di utilizzare le seguenti (sebbene vi sia un conflitto con l'abbreviazione costituita dalla combinazione del tasto Amiga sinistro con P, che sembra avere due funzioni - sembra che nessuno sia riuscito a risolvere questo dilemma, mentre la Commodore nel programma NotePad usa la combinazione del tasto Amiga sinistro e & per il paste!):

```
Amiga sinistro X - taglia (cut)
Amiga sinistro C - copia (copy)
Amiga sinistro P - incolla (paste)
Amiga sinistro I - seleziona il corsivo
Amiga sinistro B - seleziona il neretto
Amiga sinistro U - seleziona il sottolineato
Amiga sinistro P - rimette lo stile normale
Amiga sinistro Q - esegue un undo o cancella
Amiga sinistro S - salva
```

Come per le abbreviazioni, la Commodore ha specificato alcuni tasti generali da utilizzare per selezionare delle porzioni che si trovano a destra o a sinistra del cursore, a seconda del progetto, soprattutto in un word processor:

```
Amiga destro I - seleziona piccole parti alla
    destra del cursore (parole)
```

```
Amiga destro O - seleziona parti più grandi a
   destra del cursore (frasi)
Amiga destro P - seleziona parti ancora più grandi
   a destra del cursore (paragrafi)
Amiga destro J - seleziona piccole parti a
   sinistra del cursore (parole)
Amiga destro K - seleziona parti più grandi a
   sinistra del cursore (frasi)
Amiga destro L - seleziona parti ancora più grandi
   a sinistra del cursore (paragrafi)
```

La Commodore ha anche stabilito che le combinazioni tasto Amiga destro N e tasto Amiga destro M servivano a portare lo schermo Workbench dietro o davanti, e che lo stesso Intuition si sarebbe occupato di questo, per cui il software non deve usare queste combinazioni per altri scopi.

Alcune software house hanno accettato queste convenzioni . Per esempio WordPerfect usa l'abbreviazione Amiga destro I per selezionare il resto della parola che si trova a destra del cursore. Sebbene in alcune parti faccia a modo suo, si conforma alle regole generali di Intuition raccomandate dalla Commodore.

La domanda inevitabile che sorge da tutto questo è: perché è necessario adottare questi standard?

L'unica risposta a questo è che serve a rendere la vita più facile all'utente finale. Pensate a un utente finale che usa diversi pacchetti sull'amiga. Usando per esempio un word processor per la maggior parte del tempo si abituerà alle convenzioni di tastiera di quel pacchetto, ma forse gli capiterà anche di dover usare di tanto in tanto un database o un foglio elettronico. Immaginate la sua confusione se scopre che le combinazioni sulla tastiera hanno funzioni diverse dalla norma, magari così diverse che vi è persino il rischio di rovinare l'intero lavoro.

Una ragione per la quale alcune software house modificano gli standard è che il loro software è stato scritto per lavorare su di una gamma di computer i cui standard sono forse leggermente diversi (sempre che posseggano degli standard). La ragione a sostegno di questo è che l'utente che usa uno stesso programma su di una macchina qualsiasi vuole trovare lo stesso ambiente sull'Amiga. Considerando che la percentuale di tali utenti probabilmente corrisponde a meno di una frazione dell'uno per cento, ma che quasi il cento per cento degli utenti di quel programma useranno anche altri programmi sull'Amiga, la questione non è neanche da discutere. È molto più probabile che la politica interna della società abbia stabilito che è più facile produrre un solo manuale per tutte le macchine, e che personalizzare il nucleo del software di ogni computer è una perdita di tempo e denaro. Queste tesi ignorano completamente le realtà della vita, ignorano totalmente l'utente finale e, soprattutto, implicano che lo stesso software non è di alcuna utilità se non viene usato nel suo ambiente specifico.

Vi è anche un pacchetto che, persino nelle varie sezioni dello stesso programma, possiede diverse convenzioni mescolate tra loro! In alcuni casi, la combinazione Amiga sinistro X esegue il CUT (come avviene in un requester del testo fornito da Intuition) ma in altri casi la stessa combinazione non esegue il CUT del testo, tuttavia permette all'utente di cambiare le directory nelle quali sta lavorando; a questo punto per cancellare il testo bisogna usare CTRL-X! Questo è solo uno degli esempi, comunque non fa nulla per dare all'utente un po' di confidenza. Molti utenti finali probabilmente non noteranno mai le frustrazioni prodotte da tali incoerenze, ma le loro prestazioni e l'opinione che avranno del prodotto inconsciamente verranno intaccate.

Lo stesso problema succede anche con l'Amiga, poiché usando il CLI, CTRL-X ha la funzione del CUT, mentre usando uno "string gadget" viene utilizzata la combinazione L-Amiga X.

La coerenza ed il multi-tasking

In Amiga, come in tutti gli ambienti in multi-tasking, la coerenza è di vitale importanza. A livello software è NECESSARIO obbedire alle regole altrimenti i programmi in competizione per le risorse del sistema si scontreranno ed infine si bloccheranno sotto i nostri occhi. In sistemi più semplici tale coerenza non sempre è necessaria.

Anche a livello di interfaccia utente la coerenza è estremamente importante. In un sistema semplice a singolo programma, l'incoerenza non ha molta importanza poiché il passaggio da un programma all'altro richiede del tempo e l'utente apprenderà presto gli indizi relativi ad esso che lo aiuteranno ad orientarsi nell'ambiente di quel programma proprio come un guidatore di una macchina con guida a destra si può di solito abituare velocemente alla guida a sinistra, perché si trova in un ambiente diverso per un ragionevole periodo di tempo. Tuttavia, in un sistema multi-tasking, ove è possibile che un database ed un word processor lavorino allo stesso tempo, il commutare dall'uno all'altro non dà molti indizi riguardo al programma e l'utente è quasi costretto a trasferire la memoria delle combinazioni basilari oltre i confini del programma. Se tali combinazioni non sono coerenti sorgono dei problemi, come sorgerebbero per un guidatore di un'automobile con guida a destra al quale si trasferisse la guida a sinistra e viceversa ad ogni cambio di marcia!

Forse quello che voglio dire in questo articolo è che le software house dovrebbero guardare oltre i loro prodotti, verso il sistema Amiga nel suo insieme. Per favore, dimenticatevi della coerenza all'interno della vostra produzione (a meno che possa essere adattata senza interferire con le regole base dell'Amiga) e lavorate per un ambiente Amiga coerente. Sfortunatamente credo che, in alcuni pacchetti Amiga, l'arroganza e un certo grado di disprezzo per l'utente finale siano già emersi, così da rovinarli.

Questo articolo dà per scontata l'intera psicologia dell'interfaccia utente. È un argomento complesso e viene raramente capito persino dai migliori autori di software.

Continua a pagina 82

Programmare in ARexx

Parte 1 - sintassi, simboli, operatori e opzione trace

di John Carpenter

ARexx è un linguaggio di programmazione di sistema che, a parte le normali caratteristiche di un linguaggio, permette ai task di Amiga di comunicare direttamente fra di loro. Si possono costruire complesse sequenze di comando simili a quelle del CLI e un programma ARexx, o un programma applicativo che sia provvisto di interfaccia ARexx, può *pilotare* programmi esterni (incluso del software applicativo come, per esempio, SuperBase). Molti programmi commerciali sono ormai forniti di una interfaccia ARexx e il futuro sembra molto promettente.

É stato con grande eccitazione che ho letto, alcuni mesi fa, un articolo che annunciava che William Hawes aveva implementato il linguaggio Rexx su Amiga chiamandolo ARexx.

Rexx è un'abbreviazione per **Restructured Extended Executor**, un linguaggio sviluppato da Mike Cowlishaw dei laboratori IBM inglesi. In origine fu pubblicato (almeno così credo) come sostitutivo del linguaggio EXEC che girava su CMS, facente parte del sistema operativo VM dell'IBM.

L'implementazione Amiga segue fedelmente la definizione del linguaggio come descritta nel libro "The Rexx Language: A Practical Approach in Programming" di M. F. Cowlishaw, pubblicato da Prentice Hall.

Rexx, come il BASIC, è un interprete, e questo, fatta eccezione per alcune parole chiave in comune, mette fine alle similarità fra i due linguaggi. Possiede una sofisticata interfaccia per le istruzioni che gli permette di lanciare programmi esterni e di comunicare con essi, rendendo possibile al programmatore lo sviluppo di pacchetti software completamente integrati.

Rexx è molto facile da usare anche per un principiante. La sintassi del linguaggio è uniforme e non soffre delle specializzazioni di cui soffrono alcuni dialetti del BASIC. Una delle caratteristiche migliori per il principiante (e per il programmatore esperto) consiste nella superba possibilità offerta dal modo *trace* per il debugging a livello di codice sorgente.

Un programma Rexx

Un programma Rexx consiste in un testo ASCII (che può essere creato usando qualsiasi editor che produca un file ASCII come output) contenente una serie di frasi. Una frase consiste in spazi bianchi (che vengono ignorati, a meno che non fungano da delimitatori o facciano parte di una stringa), parole chiave Rexx, simboli e caratteri speciali. Le parole chiave Rexx sono nomi riservati per le istruzioni Rexx.

I **simboli** si dividono in commenti, stringhe, simboli speciali, numeri e operatori. I caratteri speciali sono ( ) , ; : e hanno un significato particolare quando sono al di fuori di una stringa.

Una **frase** è delimitata da un carattere di new-line, eccetto quando l'ultimo carattere nella linea è una virgola (fatta eccezione per i commenti), o quando la linea termina nel bel mezzo di una stringa o di un commento. La frase, che d'ora in poi chiameremo linea di istruzione, può essere esplicitamente terminata utilizzando il punto e virgola (;).

Gli **spazi bianchi**, a eccezione di quelli contenuti nelle stringhe, sono sempre ridotti a un singolo carattere di spaziatura, mentre le linee vuote vengono ignorate. Dopo questa compressione, la linea di istruzione viene letta da sinistra a destra dall'interprete.

I **commenti** consistono in una qualsiasi sequenza di caratteri delimitata da /* e da */ e possono riempire più di una linea. Possono anche essere inseriti in mezzo ad altri simboli. I commenti vengono ignorati dall'interprete che li tratta come separatori e li converte quindi in un singolo spazio.

I programmi Rexx *devono* iniziare con un commento. Questo è stato fatto per mantenere la compatibilità fra i programmi Rexx e i programmi EXEC dell'ambiente CMS.

Le stringhe

Una stringa è una sequenza di caratteri da usare così come sono. Viene delimitata dall'apice (') o dai doppi apici ("). Possono essere usate anche stringhe in binario o in esadecimale.

```
""          /* stringa nulla o vuota */
"abc"       /* stringa contenente abc */
'abc'       /* stessa stringa delimitata
              diversamente */
```

```
'John''s book' /* John's book. I due apici
                  indicano un apice singolo */
"John's book"  /* equivalente al precedente */
"lui disse, ""aiuto""" /* diventa: lui
                  disse, "aiuto" */
'12AB'x        /* stringa esadecimale */
"12 ab"x       /* equivalente al precedente */
'00000001'b    /* stringa binaria */
```

I simboli

I simboli iniziano con un carattere alfabetico o con un carattere speciale, ma non possono cominciare con un punto (.), con un punto e virgola (;) o con un due punti (:). Questi simboli possono essere usati come variabili, come etichette e, se non vengono inizializzati, possono essere anche trattati come stringhe.

La parte rimanente di un simbolo può contenere numeri e alcuni caratteri speciali. Tutte le lettere di un simbolo vengono automaticamente trasformate nelle corrispondenti maiuscole prima dell'esecuzione.

Quando un simbolo non viene inizializzato e non funge da etichetta assume il valore del proprio nome, sebbene, per esperienza personale, è meglio non fare uso di questa caratteristica del linguaggio, se non in programmi molto semplici.

Un'etichetta è un simbolo terminante con un due punti (:).

```
a=1     /* a è una semplice variabile */
a=b     /* b è una semplice variabile,
           eccetto quando non sia stata ancora
           inizializzata, nel qual caso assume
           il valore di b */
Tasso_interessi=0.5 /* rende i nomi delle
           variabili più leggibili */
#a=0    /* si possono usare alcuni simboli
           speciali per cominciare il nome
           di una variabile */
Call Start /* questa è una chiamata alla
           routine etichettata come Start */
Start:  /* etichetta della routine Start */
```

Gli array

Il punto (.) ha un significato speciale, in quanto viene utilizzato per creare simboli composti e radici. Un simbolo composto consiste della radice e di una o più desinenze. Supponiamo di avere un semplice array di elementi dallo 0 al 9 chiamato a. I nomi degli elementi saranno allora a.0 a.1 e così via. Possiamo utilizzare la variabile b per accedere a questi elementi, scrivendo a.b. Mettendo in b un valore compreso fra 0 e 9, otterremo il valore dell'elemento desiderato. Mettendo in b il valore 10, ci verrà restituito a.10. Si possono creare anche array multidimensionali, utilizzando più desinenze come in a.b.c.

Una radice ha solo un punto e questo deve essere l'ultimo carattere del suo nome. Questa viene utilizzata per inizializzare tutte le possibili variabili composte che hanno la stessa radice. Eseguendo a.=0 si hanno degli effetti sia in a.b, sia in a.b.c.

Se la seguente parte di programma Rexx fosse eseguita:

```
a=3    /* assegna 3 alla variabile A */
b=1    /* assegna 1 alla variabile B (notare
          che gli spazi vengono eliminati prima
          dell'esecuzione) */
c.a.b = 'Amiga'  /* assegna "Amiga"
                    all'elemento C.3.1 */
d='John'
e.john = 'Fred'  /* assegna "Fred" a E.JOHN */
Say c.3.1 e.d    /* stampa C.3.1 e E.JOHN */
```

si otterrebbe *Amiga Fred* come output.

I numeri

I numeri consistono in sequenze di caratteri numerici che possono essere precedute dal segno meno o dal segno più. I numeri possono contenere anche il punto per rappresentare il punto decimale e/o E (oppure e) per rappresentare la notazione esponenziale.

I seguenti sono tutti numeri legali in Rexx:

```
1      1.0     -1.0
2e5    2e-3    +12.34
```

Gli operatori

Gli operatori consistono nella tipica serie di operatori dei computer, includendo + - < <= e così via. La serie completa e le relative priorità si trovano nel manuale di ARexx.

Vale la pena, comunque, menzionare due operatori speciali per stringhe. Il primo consiste in due barre verticali (||) e il secondo nel semplice carattere di spazio.

Se eseguiamo la seguente istruzione:

```
a='Hello there' 'world'
```

allora *a* conterrà "Hello there world", ma scrivendo:

```
a = 'Hello there'||'world'
```

a conterrà "Hello thereworld" (nessuno spazio fra *there* e *world*).

Tutte le regole sopracitate potrebbero sembrare complesse, ma quando si comincia a utilizzare il linguaggio si scopre che la maggior parte di esse sono naturali e che viene spontaneo impiegarle.

L'opzione di trace

Per dimostrare la potenza dell'opzione di *trace* di Rexx, inserite il seguente programma in un file di testo (utilizzando qualsiasi editor che produca un output ASCII) e chiamatelo SIMPLE, ricordando che tutti i programmi Rexx devono cominciare con un commento:

```
/* Un semplice (SIMPLE) programma Rexx */
Trace Value 'Off'  /* disabilita il trace */
a=1                /* assegna 1 ad a */
b=/* possiamo mettere un commento qui */
                                    'John'
                   /* assegna "John" a b */
Say "Say" b a      /* stampa a e b */
Exit 0             /* Il comando exit è uno
                     dei modi per terminare
                     un programma Rexx */
```

Con la versione Amiga di Rexx, avremo prima bisogno di avere in esecuzione il programma principale di background, chiamato *rexxmast*. Una volta che questo è stato lanciato (scrivendo semplicemente *rexxmast* nel CLI) il programma SIMPLE può essere eseguito scrivendo:

```
RX SIMPLE
```

Il risultato di questo programma sarà:

```
Say John 1
```

Non c'è niente di difficile fino a questo punto. Adesso cambiamo la linea di trace in questo modo:

```
Trace Value 'R'  /* visualizza i risultati */
```

L'output adesso sarà:

```
    3 *-* a=1;
      >>> "1"
    4 *-* b= 'John';
      >>> "John"
    5 *-* ;
    6 *-* Say "Say" b a;
      >>> "Say John 1"
  Say John 1
    8 *-* Exit 0;
      >>> "0"
```

I numeri sulla sinistra rappresentano il numero della linea che contiene le istruzioni nel file. Il simbolo *-* indica la presenza di una linea di istruzione. Segue l'istruzione e il simbolo >>> che indica il risultato dell'operazione.

Adesso cambiamo la linea di trace in questo modo:

```
Trace Value 'I'  /* visualizza i risultati
                    intermedi */
```

L'output diventa:

```
    3 *-* a=1;
      >L> "1"
    4 *-* b= 'John';
    5 *-* ;
    6 *-* Say "Say" b a;
      >V> "John"
      >V> "1"
      >O> "Say John"
      >O> "Say John 1"
  Say John 1
    8 *-* Exit 0;
      >L> "0"
```

A questo punto, i commenti e gli spazi bianchi superflui sono stati eliminati ed è stato aggiunto il delimitatore di istruzione (;). Il simbolo >L> indica l'elaborazione di una stringa (L sta per literal). Il simbolo >V> contraddistingue il contenuto di una variabile e il simbolo >O> produce il risultato intermedio mentre l'interprete sta lavorando su due operandi alla volta. Ci sono molte altre opzioni di trace ma io tendo a utilizzare quelle sopra menzionate, o l'opzione 'ALL' che visualizza semplicemente l'intera istruzione, oppure 'OFF' che disabilita il trace.

Ci sono anche due caratteri speciali che possono essere utilizzati come prefisso per l'opzione di trace: ? e !. Il prefisso ? abilita o disabilita il trace interattivo. Premendo enter senza input viene eseguita la prossima istruzione, ma se inseriamo dei dati, questi vengono interpretati come un'istruzione a sé stante. Questo è uno strumento di debugging molto potente e personalmente tendo a utilizzarlo con le opzioni di trace *result* (R) e *intermediate* (I). Una volta provato, dubito che vorrete mai più usare un interprete di qualsiasi tipo che non disponga di questo livello di trace.

Il prefisso ! abilita o disabilita la funzione che blocca la spedizione di un comando a un programma host. Quando questa funzione è inserita, l'istruzione viene esaminata ma il comando non viene mandato al programma esterno e la speciale variabile di ritorno (RC) viene messa a zero. Il blocco della spedizione dei comandi al programma host viene rimosso quando si esegue il comando di trace specificando l'opzione OFF.

Gli assegnamenti

Nell'ultimo programma abbiamo visto l'assegnamento di stringhe e di numeri. Quelli che hanno familiarità con il BASIC si chiederanno che fine abbia fatto il suffisso $ per le variabili stringa. La risposta è che non esistono più, ma l'idea di variabili numeriche e di stringa continua a esistere. Provate ora a digitare ed eseguire questo programma:

```
/* Programma 2 */
Trace 'I'      /* trace interattivo */
a=1            /* assegna il numero 1 ad a */
b='1'          /* assegna il carattere 1 a b */
c=a+b          /* poi li somma insieme */
Say c          /* e stampa il risultato */
d='2+2'        /* mette in d una stringa
                  non-numerica */
Say c+d
Exit 0
```

Il risultato sarà:

```
    3 *-* a=1;
      >L> "1"
    4 *-* b='1';
```

```
   5 *-* c=a+b;
     >V> "1"
     >V> "1"
     >O> "2"
   6 *-* Say c;
     >V> "2"
2
   7 *-* d='2+2';
   8 *-* Say c+d;
     >V> "2"
     >V> "2+2"
+++ Error 47 in line 8: Arithmetic conversion
    error
Command returned 10/47: Arithmetic conversion
    error
```

La cosa sorprendente per i programmatori BASIC è che, nonostante b contenesse un carattere, ARexx è stato in grado di interpretarlo correttamente come un numero e di fare le conversioni necessarie alla linea 5. La stringa d non contiene, invece, un numero valido, quindi l'addizione di linea 8 non può essere fatta. Cambiando la linea 7 in questo modo:

```
d='2.2'
```

oppure in

```
d='2e2'
```

il programma funzionerà correttamente.

Ancora array

Il programma seguente esplora i simboli composti e le radici.

```
/* Programma 3 */
Trace 'R'       /* trace dei risultati */
Say 'Inserite alcune parole e premete enter'
Pull invar      /* assegna l'input a INVAR */
Do i = 0 to Words(invar)-1
                /* loop per preparare l'array */
a.i = Word(invar,i+1)
                /* assegna una parola a ogni
                  elemento */
end             /* termine del loop */
Say a.5
Exit 0
```

Lanciamo il programma e scriviamo *Hello world* quando ci verrà richiesto. Questo sarà il risultato:

```
4 *-* Say 'Inserite alcune parole e premete
          enter';
>>> "Inserite alcune parole e premete enter"
```

Inserite alcune parole, poi premete enter

```
   6 *-* Pull invar;
     >>> "HELLO WORLD"
   7 *-* Do i = 0 to Words(invar)-1;
     >>> "0"
     >>> "HELLO WORLD"
     >>> "1"
   8 *-* a.i = Word(invar,i+1);
     >>> "HELLO WORLD"
     >>> "1"
     >>> "HELLO"
   9 *-* end;
   7 *-* Do i = 0 to Words(invar)-1;
     >>> "1"
   8 *-* a.i = Word(invar,i+1);
     >>> "HELLO WORLD"
     >>> "2"
     >>> "WORLD"
   9 *-* end;
   7 *-* Do i = 0 to Words(invar)-1;
     >>> "2"
  11 *-* Say a.5;
     >>> "A.5"
A.5
  12 *-* Exit 0;
     >>> "0"
```

La linea 6 contiene l'istruzione PULL. Questa istruzione viene utilizzata per ottenere dell'input dallo stack. Quando lo stack è vuoto, come in questo caso, leggerà una stringa dalla console.

La linea 7 è l'inizio di un loop creato dall'istruzione *do* abbinata a *end* che si trova sulla linea 9. Questo uso di do-end è equivalente alle istruzioni BASIC:

```
FOR I=1 to N
NEXT
```

Alla linea 7 troviamo la funzione *Words*(). ARexx, come avrete occasione di apprendere, ha molte funzioni. Questa restituisce il numero di parole contenute in una stringa.

La linea 8 contiene l'assegnamento dell'array a. Dopo il loop *do*, l'elemento a.0 conterrà HELLO, mentre l'elemento a.1 conterrà THERE. La funzione *Word*() restituisce l'ennesima parola di una stringa.

La linea 11 stampa il sesto elemento dell'array a, ma avendo inserito solo due elementi, l'elemento 6 risulta non inizializzato. In questo caso, come abbiamo già visto, le regole stabiliscono che una variabile non inizializzata contiene il valore del proprio nome. Questo spiega per quale motivo è stato stampato A.5.

Inserendo una nuova linea fra la numero 6 e la numero 7, chiamiamola 6bis, contenente il comando:

```
a. = ""    /* inizializza l'array */
```

il programma girerà correttamente. La linea 11 stamperà adesso una linea vuota se non digitiamo la sesta parola.

Nella prossima puntata daremo un'occhiata ad alcune altre istruzioni ARexx, compresa la do-end in dettaglio.

I dischi di Fred Fish - 2

Prima guida ragionata al loro contenuto

EDITOR DI TESTI

Disco	Nome	Versione		Descrizione	Autore
95	CYGNUSEDDEMO		*	DEMO DI UN TEXT EDITOR-MASSIMO FILE DA 5k	[CYGNUSSOFT]
59	DME	1.22		EDITOR DI TESTI PER PROGRAMMATORI WYSYWYG	[DILLON]
74	DME	1.25		EDITOR DI TESTI PER PROGRAMMATORI WISIWIG	[DILLON]
87	DME	1.27		EDITOR DI TESTI PER PROGRAMMATORI WISIWIG	[DILLON]
93	DME	1.27	*	EDITOR DI TESTI PER PROGRAMMATORI WISIWIG	[DILLON]
113	DME	1.28F	*	EDITOR DI TESTI PER PROGRAMMATORI WISIWIG	[DILLON]
134	DME	1.29	*	EDITOR DI TESTI PER PROGRAMMATORI WISIWIG	[DILLON]
84	ED		*	SIMILE AS ED PER UNIX - EDITOR DI TESTI	[BEATTIE]
22	LEMACS	3.6	*	EDITOR DI TESTI	[LAWRANCE (CONROY)]
60	MED	2.1		EDITA SU 36 LINEE SIMULTANEAMENTE (USARE MOUSE)	[ROUAIX]
2	MICROEMACS	2.0	*	VERSIONE DI EMACS TEXT ED - CARATTERISTICHE LIMITATE	[CONROY/JONES]
6	MICROEMACS	2.0	*	EDITOR DI TESTI EMACS - INC SUPPORTO TASTI FUNZIONE	[ROOSE (CONROY)]
61	MICROEMACS	3.8B	*	EDITOR DI TESTI	LAWRENCE (CONROY)]
93	MICROEMACS	3.8I	*	EDITOR DI TESTI	[LAWRENCE (CONROY)]
119	MICROEMACS	3.9E	*	EDITOR DI TESTI	[LAWRENCE (CONROY)]
23	MICROEMACS	30	*	POTENTE EDITOR DI TESTI	
42	MICROGNUEMACS	1A	*	PICCOLO E POTENTE EDITOR DI TESTI	
131	MICROGNUEMACS	1B	*	PICCOLO,POTENTE EDITOR (MACROS E PORTA AREXX)	[ROKICKI]
22	PEMACS		*	EDITOR DI TESTI	[POGGIO (CONROY)]
20	TED	0.85		VERSIONE DEMO DELL'EDITOR DI TESTI	[HEATH]
83	TEX			VERSIONE DEMO DI UN EDITOR DI TESTI / PER PICCOLI FILES	[ROKICKI]
31	TXED	1.1		MICROSMITH EDITOR-DEMO (MASSIMO FILE 10K)	[HEATH]
60	UEDIT	2.0		POTENTE EDITOR DI TESTI / CONSIGLIABILE	[STYLES]
121	UEDIT	2.3		EDITOR DI TESTI	[STILES]
60	UETURBO			VERSIONE PER PROGRAMMATORI DI UEDIT	[ALTHOFF]

FORMATTATORE DI TESTI

Disco	Nome	Versione		Descrizione	Autore
65	BAWK		*	PROCESSORE DI TESTO (ALA UNIX AWK)	[BRODT]
92	BAWK		*	PROCESSORE DI TESTO (ALA UNIX AWK)	[WIDEN (BRODT)]
14	DEX		*	ESTRAE COMMENTI DA UN FILE SORGENTE IN FORMATO NROFF	[FISH]
3	FF		*	FORMATTATORE DI TESTO MOLTO VELOCE	[PERMAN]
79	NRO		*	PROCESSORE ED EDITOR DI TESTO STILE ROFF	[BROWING]
9	PROFF	1.1	*	POTENETE FORMATTATORE TESO -RISCRITTURA DI ROFF PER UNIX	[YIGIT/TRESS]
46	PROFFMACROS		*	SETS DI MACROS PER ROFF	[ANDREWS/ WALKER]
3	ROFF		*	FORMATTATORE TESTO (COME LA VERSIONE DI SOFTWARE TOOLS)	[YAP]
128	SED		*	EDITOR A SCORRIMENTO UNIX - TEXT EDITOR	[RAYMOND]

GIOCHI DI RICREAZIONE

Disco	Nome	Versione		Descrizione	Autore
120	AMOEBA			CLONE DI SPACE INVADERS	[LATENIGHT DEV]
122	ASTEROIDS			GIOCO TIPO ASTEROIDS - SUONO E IMMAGINI SOSTITUIBILI	[MARIANI]
28	BACKGAMMON		*	GIOCO GRAFICO DEL BACKGAMMON (ABASIC)	[ADDISON]
120	BACKGAMMON	1.0	*	GIOCO GRAFICO DEL BACKGAMMON	[PFISTER]

Disco	Nome	Versione		Descrizione	Autore
19	BLACKJACK		*	GIOCO DI CARTE DEL BLACKJACK / VERSIONE TESTO	[PLUM HALL INC.]
50	BREAKOUT	1.0		VERSIONE 3D DEL GIOCO TRADIZIONALE - CON OCCHIALI 3D	[KEMP]
13	BRICKOUT.BAS		*	GIOCO BRICKOUT - USA IL MOUSE COME CONTROLLORE (ABASIC)	[DAVISON]
32	CANFIELD		*	GIOCO DI CARTE (ABASIC)	[ADDISON]
96	CHESS	1.0	*	BUON GIOCO DEGLI SCACCHI - NON AMIGA'D	[LEIVIAN (STANBACK)]
45	CLUE		*	CLUEDO STILE GIOCO DA TAVOLO	[PRYOR]
10	CONQUEST	1.0		CONTROLLO DI UNA IMPERO INTERSTELLARE	[SHIMBO]
24	CONQUEST	1.1	*	AVVENTURA SPAZIALE	[SHIMBO]
51	COS			RUOTA DELLA FORTUNA - (AMIGABASIC)	[MICHEL]
40	COSMO			SEMPLICE ARCADE STILE SPACE SHOOT GAME	[HARRIS]
28	CRIBBAGE		*	GIOCO DI CARTE (ABASIC)	[ADDISON]
78	CYCLES	1.0		ISPIRATO AL FILM TRON	[GILMORE]
120	EGYPTIANRUN	1.1		SEMPLICE GIOCO DI CORSA E D'AZZARDO	[HAMES]
118	EMPIRE	1.0	*	SIMULAZIONE POLITICA,ECONOMICA E DI GUERRA PER + GIOCAT.	[GRAY/ LANGSTON]
78	EOMS			GIOCO DI SIMULAZIONE MERCENARIA PER ESPERTI	[CARDENAS]
70	GRAVITYWARS	1.03		GIOCO NAVE/MISSILE CON GRAVITA'	[BARZ]
84	GRAVITYWARS	1.04	*	GIOCO NAVE/MISSILE CON GRAVITA'	[BARTZ]
105	GRAVITYWARS	2.0		GIOCO NAVE/MISSILE CON GRAVITA'	[BARTZ]
7	HACK	1.0.1		GIOCO D'AVVENTURA HACK (DATI SORGENTI SU FISH 8)	[TOEBES]
32	KLONDIKE		*	GIOCO DI CARTE - (AMIGABASIC)	[ADDISON]
63	LARN	12.0B		GIOCO D'AVVENTURA/LABIRINTO (COME HACK)	[BURNETTE (MORGAN)]
28	MILESTONE	1.0	*	CORSA IN AUTOMOBILE (ABASIC)	[ADDISON]
50	MISSILE	1.1		GIOCO DIFESA MISSILE	[MERRIMAN]
15	MONOPOLY			GIOCO DA TAVOLO MONOPOLY (ABASIC)	[ADDISON]
28	OTHELLO		*	GIOCO DELL'OTHELLO (ABASIC)	[ADDISON]
90	OTHELLO	1.0		GIOCO DELL'OTHELLO	[BELLEW (ROY)]
122	PUSHOVER		*	GIOCO TIPO FORZA 5 (AMIGA BASIC)	[YOST]
32	PUZZLE	1.2		CLASSICO PUZZLE A 15 PEZZI	[BEOGELEIN]
38	REVERSI	6.1		GIOCO DELL'OTHELLO	[ALMUDEVAR]
106	RISTINOLLA	1.0		VERSIONE FINALE DI GO-MOKU	[PIHLAJAMAKI]
82	ROCKET			MISSILE CHE ATTERRA SU UNA FINESTRA	[DA SILVA]
85	ROCKET		*	MISSILE CHE ATTERRA SU UNA FINESTRA	[DA SILVA]
55	SHANGHAIDEMO			GIOCO DELL'ACTIVISION BASATO SU MAHJONG - DEMO	[ACTIVISION]
103	SOL		*	SOLITARIO	[GOODENOUGH/ SWANK]
10	TREK73		*	GIOCO DI STAR TREK - BASATO SUL TESTO	
35	TRICLOPS	1.6		GIOCO SPAZIALE 3D	[GEODESICPUBL]
36	TUNNELVISION		*L	LABIRINTO - CON VEDUTE 3D DALL'INTERNO IN PROSPETTIVA	[ADDISON]
100	WBLANDER	2.1		SEMPLICE ATTERRAGGIO SULLA LUNA	[DA SILVA]
114	WLANDER	2.1	*	SEMPLICE ATTERRAGGIO SULLA LUNA - CON SORPRESA	[DA SILVA/ LEHENBAUER]
10	YACHTC		*	GIOCO DELLO YAHTZEE DICE	[LEEMON]

LETTORE DI TESTI

Disco	Nome	Versione		Descrizione	Autore
60	BLITZ	1.0		LETTORE DI FILES DI TESTO VELOCE-ATTIVABILE DA HOY-KEY	[HAUGEN]
34	LESS		*	LETTORE DI TESTO CON MOVIMENTI AVANTI/DIETRO	[NUDELMAN/ LEIVIAN]
74	LESS	1.1	*	LETTORE DI FILES DI TESTO - MUOVE AVANTI/DIETRO	[LEIVIAN (NUDELMAN]
36	LEX			COMPUTA INDICI LEGGIBILI DI FILES TESTO	[SULLIVAN]

MONITOR PER IL SISTEMA

Disco	Nome	Versione		Descrizione	Autore
40	AMIGAMONITOR	0.21		MOSTRA FILES APERTI,MEMORIA,TASK ECC IN TEMPO REALE	[VORIS]
70	AMIGAMONITOR	1.13		MOSTRA FILES APERTI,MEMORIA,TASK ECC IN TEMPO REALE	[VORIS]
111	AMYLOAD		*	MONITOR GRAFICO DI CPU,BLITTER E USO MEMORIA	[KELLY]
5	FREEMAP		*	VISUALIZZA DIAGRAMMI DELLA MEMORIA LIBERA	[MIKAL]
111	GAUGE			VISUALIZZA LA MEMORIA CON BARRE GRAFICHE VERTICALI	[DA SILVA]
1	GFXMEM	0.4	*	MOSTRA L'USO DELLA MEMORIA DI AMIGA GRAFICAMENTE	[MAMAKOS]
14	GFXMEM	0.5	*	MOSTRA L'USO DELLA MEMORIA DI AMIGA GRAFICAMENTE	[MAMAKOS]
54	LAV		*	MOSTRA NUM. DEI TASK IN CODA D'ESECUZIONE NEL TITLE BAR	[ROCKIIDGE]
123	LIBS			VISUALIZZA LISTA LIBRERIE E DETTAGLI DISPONIBILE	
108	MONIDCMP		*	MONITOR INTUIMESSAGES PASSANDO ATTRAVERSO IDCMP	[CERVONE]
69	MONPROC	1.0	*	MONITOR AMIGADOS PROCESSI PACCHETTI ATTIVI	[LINSAY]
79	MONPROC	1.1	*	MONITOR PACCHETTI ATTIVI	[CERVONE (LINDSAY)]
69	SB	1.0	*	SCORRE LE STRUTTURE SISTEMA	[SULLIVAN/ ZAMARA]

58	SYSMON			MONITOR GRAFICO DI VARI PARAMETRI DI SISTEMA	[KIVOLOVITS]
74	TDEBUG	1.00	*	MONITOR DEVICE IO REQUEST PER DEBUGGING	[DILLON]
79	WHO		*	LISTA DEI TASK NELLA CODA PRONTI/ATTESA	[MUSSER]
73	XPLOR		*	MOSTRA SOMMARIAMENTE TUTTE LISTE SISTEMA DA EXECBASE	[PHILIPS]

PROGRAMMI DI VARIO TIPO

33	3DSTARS		*	CAMPO DI STELLE 3D (CI VOGLIONO OCCHIALI ROSSI/BLU 3D)	[SCHWAB]
15	BLOBS		*	DISEGNA LINEE VAGANTI COLORATE	[ENGELBRITE]
71	BLOCKS			VISUALIZZA QUADRATI COLORATI IN MOVIMENTO (COME LINEE)	[WALKER]
127	BOUNCE		*	CREA PUNTI CHE RIMBALZANO ATTORNO E MULTIPLI	[HANSEL/ HANSEL]
126	DANCE1		*	MOVIMENTO POLIGONI	[OLSEN]
126	DANCE2		*	MOVIMENTO POLIGONI IN MODO HAM	[OLSEN]
15	DAZZLE		*	DISEGNA PUNTI CASUALI,MA CON 8-PIEGHE SIMMETRICHE	[ENGELBRITE]
89	DEMOLITION		*	PUNTO CHE RIMBALZA ATTORNO SCHERMO-MANGIANDOSI NEL PERC.	[KIRYMIS]
66	DK	1.0	*	VISUALIZZA LENTA DISINTEGRAZIONE	[HANDEL]
69	DK	1.1	*	VISUALIZZA LENTA DISINTEGRAZIONE	[HANDEL]
1	DOTTY		*	SORGENTE PER DEMO SUL DISCO DEL WORKBENCH	[LUCK]
105	DRUNKENMOUSE		*	FA MUOVERE IL PUNTATORE DEL MOUSE COME SE TU HAI BEVUTO	[LIVSHITS]
66	FLIP		*	GIRA LO SCHERMO DALL'ALTO AL BASSO	[BERRO]
52	HAMPOLY		*	DISEGNA POLIGONI MULTIPLI - IN MODO HAM	[HOLSEN]
54	ING		*	FA RIMBALZARE LE WINDOW ATTORNO ALLO SCHERMO	[SCHWAB]
46	JIVE		*	SCRIVE IN TESTO CHIARO E PRODUCE JIVE TALK	
41	LINES		*	DISEGNA LINEE CHE SI MUOVONO	[JATKOWSKI]
66	MELT		*	SCHERMO CHE SI SCIOGLIE VERSO IL BASSO	[COY]
9	MOIRE		*	CREA DISEGNI UTILIZZANDO ALGORITMI PREDEFINITI	[BALLANTYNE]
87	MINCHINSQ		*	GENERA QUADRATI BASATI SU PATTERN DIPENDENTI DAI PRECED.	[SCHWAB]
137	MUNCHO			GULP QUANDO DISCO E' INSERITO(E YUCH QUANDO E' RIMOSSO)	[WERTH]
66	NART		*	MOVIMENTO DI LINEE GRAFICHE	[SCHWAB]
90	NEWDEMOS			NUOVA VERSIONE LINEE E QUADRATI-UTILIZZA MENO TEMPO CPU	[KOREN]
33	OING		*	WINDOW PIENE DI PALLE RIMBALZANTI	[SCHWAB]
18	PIGLATIN	1	*	TRADUCE DA INGLESE A "LATINO GREZZO"	[CLEMENT]
49	POLYGON		*	DISEGNA ROMBI E CERCHI CONCENTRICI SOVRAPPONENDOLI	[GINTZ]
90	RAINBENCH			FAI CICLARE I COLORI DEL WORKBENCH	[HODGSON]
58	RAINBOW		*	ASSEGNA AL BACKGROUND COLORI SIMILE A QUELLI DI MARAUDER	[HODGSON]
59	ROBOTROFF		*	ROBOT CHE SEGUE IL PUNTATORE DEL MOUSE	[SCHWAB]
82	SAND			GRANELLI DI SABBIA CHE SEGUONO PUNTATORE QUANDO SI MUOVE	[VAUGHAN]
81	SCAT			VISUALIZZA HACK-ATTIVA WINDOW SOTTO PUNTATORE	
90	SIZZLERS	1.7.0		GENERA UNA SERIE DI LINEE IN MOVIMENTO DEMO	[EPLEY]
50	SIZZLERS	1.0.0		GENERA UNA SERIE DI LINEE IN MOVIMENTO DEMO	[EPLEY]
81	SMOSH			VISUALIZZA HACK-CI VUOLE FILE IFF PER VEDERE COSA ACCADE	[ORRIS]
9	SPARKS		*	LINEE CHE CAMMINANO-SEMPLICE DEMO GRAFICO	[BALLANRYNE]
33	SPROING		*	PALLE RIMBALZANTI-COLLISIONI COL BORDO(+SUONO)	[SCHWAB]
33	STARS		*	VISUALIZZA UNO SCHERMO PIENO DI STELLE	[SCHWAB]
118	STARS			NAVE SPAZIALE CHE VIAGGIA ATTRAVERSO UN CAMPO STELLATO	[ORRIS]
43	STEMULATOR			SCHERMO E RUMORI IN STILE ST QUANDO FAI QUALCOSA	[ADDISON]
81	TARGET	1.23		TRASFORMA PUNTATORE MOUSE IN MIRINO CON UN COLPO FUCILE!	
54	TILT		*	INCLINA LO SCHERMO IN SU DA UN LATO	[SCHWAB]
113	TILT		*	INCLINA LO SCHRMO IN SU DA UN LATO	[SHAUB (SCHWAB)]
32	TRAILS		*	LASCIA UNA SCIA DIETRO IL MOUSE	[BIELAK]
84	VIACOM		*	VISUALIZZA HACK-CREA UNA FIGURA RUMOROSA	[SCHWAB]
112	VIACOM			VISUALIZZA HACK-CREA UNA FIGURA RUMOROSA	[NESBITT (SCHWAB)]
112	WAVEBENCH		*	INCRESPA LO SCHERMO DEL WORKBENCH	[NESBITT]
36	YABOING			AFFERRA SPRITES CON IL MOUSE (SCOPRE COLLISIONI DEMO)	
136	YABOING	II	*	AFFERRA SPRITES CON IL MOUSE (SCOPRE COLLISIONI DEMO)	

SHELL

18	ASH			PREREALIZZAZIONE SHELL-STORI,LOOPS,SOSTITUZIONI	[SMITH]
24	CSH	1.1	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DILLON]
36	CSH	2.01A	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DILLON]
41	CSH	2.03		SOSTITUZIONE SHELL STILE CSH PER CLI	[DILLON]
48	CSH	2.04	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DILLON]
48	CSH	2.04M	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DREW (DILLON)]
55	CSH	2.05M	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DREW (FRLOM DILLOM)]
85	CSH	2.06M	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DREW (DILLON)]

Disco	Programma	Versione		Descrizione	Autore
107	CSH	2.07M	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DREW (DILLON)]
70	JOBS	2.1		SVILUPPO ALTERNATIVO PER WB E CLI	[SAWAYA]
38	JSH		*	SEMPLICI COMANDI INTERPRETE LINEA	[KENT]
4	MYCLI		*	SOSTITUZIONE CLI	[SCHWARZ]
14	SHELL	1.0	*	SOSTITUZIONE SHELL STILE CSH PER CLI	[DILLON]

SISTEMA

Disco	Programma	Versione		Descrizione	Autore
53	ARP		*	PROGETTO SOSTITUZIONE DEI COMANDI AMIGADOS	
123	ARP	1.0		PROGETTO SOSTITUZIONE AMIGADOS /VERSIONE PIU' AGGIORNATA	
87	ID-HANDLER	1.0	*	DEVICE HANDLER AMIGADOS CON IDENTIFICATORI UNICI	[PUCKETT]

SPREADSHEET

Disco	Programma	Versione		Descrizione	Autore
104	ANALYTICALC		*	SPREADSHEET A PIENE CARATTERISTICHE	[EVERHART]
36	VC	1.0		VISICALC COME SPREADSHEET	[HARDIE (GOSLING)]
53	VC	1.1	*	VISICALC COME SPREADSHEET	[BOND (GOSLING)]

TERMINALI PER COMUNICAZIONI

Disco	Programma	Versione		Descrizione	Autore
98	ACCESS	0.18		80X25 / XMODEM / WXMODEM / MULTICOLORE	[YOUNG (JAMES)]
40	AHOST	0.9		EMULATORE DI TERMINALE ANSI	[WILHITE/ JONES]
82	AMICTERM	.50		80X25 / X/WX/ZMODEM	[SALAS/KIRK (JAMES)]
18	AMIGADISPLAY		*	80X24 / ASCI DOWNLOAD	[WOODS (MOUNIER)]
1	AMIGATERM		*	77X23 / XMODEM & TRASFERIMENTO ASCII	[MOUNIER]
12	ARGOTERM	0.20	*	77X23 / XMODEM & TRASFERIMENTO ASCII	[SAN]
26	C-KERMIT		*	TERMINALE VIRTUALE KERMIT & TRASFERIMENTO FILE	[CERVONE (MANY)]
48	COMM	1.30		EMULATORE VT100	[JAMES]
67	COMM	1.33		EMULATORE VT100	[JAMES]
71	COMM	1.34		EMULATORE VT100	[JAMES]
75	COMM	1.34	*	EMULATORE VT100	[JAMES]
40	DG210			EMULATORE DI DATA GENERAL D-210	[LENZ]
73	DTERM	1.10		TEMINALE / XMODEM / KEY & MAPPA CARATTERI	[DILLON]
60	HANDSHAKE	1.20A		COMPLETO EMULATORE V752 / VT100 / VT102	[HEBERFELLNER]
4	KERMIT	1.1	*	VECCHIA VERSIONE DI KERMIT	
14	PDTERM	1.21	*	80X25 TERMINALE VT100 / ANSI / NON TRASFERISCE	[MCLNERNY]
20	SPEECHTERM		*	TERMINALE ANSI CON XMODEM. / SINTESI VOCALE & GRAFICI	[KOUTSOFIOS]
12	STARTERM	1.1		TERMINALE CON ASCII / XMODEM / TASTI FUNZIONE	[NANGANO/ PLEGGE]
30	STARTERM	3.0		TERMINALE / TRASFERIMENTO XMODEM	[NANGANO]
108	TEK	2.65	*	EMULATORE VT100 & TEKTRONIX 4010/4014	[GIORDANO/ WHELAN]
33	TERMPLUS	2.1	*	RISCRITTURA DI AMIGATERM	[RAKOSKY (MOUNIER)]
52	TEX4010	2.1	*	EMULATORE TEK 4010	[WHELAN (MOUNIER)]
29	VT100	1.0	*	EMULATORE VT100 / KERMIT & XMODEM	[WECKER (MOUNIER)]
33	VT100	2.0	*	EMULATORE VT100 / KERMIT & XMODEM	[WECKER]
36	VT100	2.2	*	EMULATORE VT100 / KERMIT & XMODEM	[WECKER]
41	VT100	2.3	*	EMULATORE VT100 / KERMIT & XMODEM	[WECKER]
47	VT100	2.4	*	EMULATORE VT100 / KERMIT & XMODEM	[WECKER (MOUNIER)]
55	VT100	2.6	*	EMULATORE VT100 / KERMIT & XMODEM	[WECKER (MOUNIER)]
138	VT100	2.6	*	EMULATORE VT100 / KERMIT & XMODEM	[BARSHINGER (WECKER)]
114	VT100	2.7	*	EMULATORE VT100 / KERMIT & XMODEM	[SUMRALL (WECKER)]
138	VT100	2.8	*	EMULATORE VT100 / KERMIT & XMODEM	[SUMRALL (WECKER)]

Un abito firmato, una vacanza indimenticabile. Uno stereo tutto nuovo, un computer o l'ultimo modello di tv color

perfino una polizza assicurativa!
E tutto a prezzi esclusivi. Con la nuova, fantastica Jackson Card '90 anche questo è possibile.
Grazie a un accordo esclusivo, infatti, il titolare Jackson Card '90 ha diritto a uno sconto speciale presso tantissimi esercizi convenzionati*: American Contourella, Coeco, Commodore, Galtrucco, GBC, Jolly Hotels, Misco, SAI, Salmoiraghi Viganò e Singer.
Ma i vantaggi continuano. La nuova Jackson Card '90, offre anche:

- sconto speciale del 10% sull'acquisto di libri Jackson;
- invio gratuito della rivista Jackson Preview Magazine per tutto l'anno;
- invio gratuito del catalogo libri Jackson;
- speciale buono da 15.000 lire sul primo ordine di libri Jackson effettuato per corrispondenza direttamente presso l'editore, e negli stand Jackson in tutte le fiere specializzate.

E avere Jackson Card '90 é facile: basta abbonarsi o rinnovare il proprio abbonamento a una delle riviste del Gruppo Editoriale Jackson, acquistare libri Jackson per almeno 100.000 lire nelle librerie e computershop convenzionati in tutta Italia o ordinarli direttamente dall'editore.

Jackson Card '90: nuova, più ricca, sempre più preziosa.

* Tutti gli indirizzi sono pubblicati su Jackson Preview Magazine.

Indirizzamento indiretto da registro

Accedere alle strutture dati in assembly è facile come in C!

di Amanda Jones

Una gran parte del sistema operativo di Amiga si basa su raggruppamenti di dati (strutture) collegati fra di loro da puntatori. Il linguaggio C si presta molto bene alla manipolazione di queste complesse strutture dati, ma anche il microprocessore 68000 è particolarmente portato per effettuare questo tipo di lavoro. È un fatto abbastanza sorprendente, ma il codice necessario per accedere ai campi dati, contenuti all'interno di agglomerati di variabili, come le *strutture* del C, non è più difficile da scrivere in assembly piuttosto che in C.

Le definizioni e i flag utilizzati dai programmatori C hanno delle controparti nelle definizioni che troviamo nei file include (.i) di Amiga. Le variabili complesse, definite come strutture nei file header (.h) del C, hanno degli equivalenti in questi file include, in modo che ogni membro di una struttura sia descritto in modo unico da un nome e da un offset numerico rispetto a un indirizzo di base. Questi offset sono forniti per permettere al programmatore assembly di usare un modo di indirizzamento del 68000 conosciuto come *indiretto da registro con spiazzamento* (address register indirect with displacement).

Una struttura tipica

La maniera più semplice per descrivere questo modo di indirizzamento consiste nel vedere un esempio, che nel nostro caso si basa sulla struttura IntuiMessage.

Le strutture IntuiMessage sono il tramite per lo scambio di pacchetti informativi da e per le porte IDCMP delle finestre. Sono costituite da una struttura Message dell'Exec e da una parte di estensione (i cui dettagli sono forniti nel file *intuition*.h e nel manuale di *Intuition*):

```
struct IntuiMessage {

    struct Message         ExecMessage;
    ULONG                  Class;
    USHORT                 Code;
    USHORT                 Qualifier;
    APTR                   IAddress;
    SHORT                  MouseX, MouseY;
    ULONG                  Seconds, Micros;
    struct Window          *IDCMPWindow;
    struct IntuiMessage    *SpecialLink;
    }
```

Se elenchiamo il numero di byte che ogni campo di questa struttura occupa, ecco cosa troviamo:

Byte	Nome del campo
20	ExecMessage
4	Class
2	Code
2	Qualifier
4	IAddress
2	MouseX
2	MouseY
4	Seconds
4	Micros
4	*IDCMPWindow
4	*SpecialLink

Le dimensioni di questi campi possono essere utilizzate per creare una tabella di valori espressi in byte, che mostri quanto lontano dobbiamo spostarci dall'inizio di una struttura IntuiMessage per trovare un determinato campo. Il risultato sarà una collezione di offset (spiazzamenti) relativi alla base della struttura IntuiMessage:

Offset	Nome del campo
0	ExecMessage
20	Class
22	Code
24	Qualifier
28	IAddress
30	MouseX
32	MouseY
36	Seconds
40	Micros
44	*IDCMPWindow
48	*SpecialLink

Fortunatamente non è necessario calcolare questi offset in quanto sono già presenti negli appropriati file include.

I nomi dei campi di una struttura, in assembly, hanno un prefisso aggiunto rispetto ai corrispondenti usati nel C. Questo prefisso si riferisce al nome della struttura che li contiene nel C ed è stato aggiunto per evitare casi di omonimia. I nomi dei campi della struttura IntuiMessage, che troveremo nell'appropriato file include, sono precisamente questi:

Offset	Nome del campo (file .i)
0	im_ExecMessage
20	im_Class
22	im_Code
24	im_Qualifier
28	im_IAddress
30	im_MouseX
32	im_MouseY
36	im_Seconds
40	im_Micros
44	im_IDCMPWindow
48	im_SpecialLink

Poiché tutti i calcoli relativi agli offset sono stati già fatti per noi, tutto quello che ci rimane da fare è vedere come si usano.

Indirizzamento indiretto

L'indirizzamento indiretto implica che, invece di specificare un indirizzo, si specifica la *locazione* di questo indirizzo. Il processore 68000 può usare solo i registri indirizzi per calcolare l'indirizzo indiretto, da cui il nome *indiretto da registro*.

Vediamo, per prima cosa, un esempio ordinario di indirizzamento indiretto da registro. L'istruzione:

```
move.l   (a0),d2
```

utilizza l'indirizzamento indiretto da registro per specificare la locazione dell'operando sorgente; in sostanza dice che il dato deve essere letto dalla locazione *il cui indirizzo è contenuto nel registro A0* e poi deve essere copiato nel registro D2.

Adesso vediamo quello che ci serve per gestire le strutture dati in assembly. Il 68000 ci permette di specificare un valore di spiazzamento nell'istruzione appena vista:

```
move.l  im_Class(a0),d2
```

Se nel registro A0 è stato messo il puntatore a una struttura IntuiMessage (o, detto in altre parole, se A0 contiene l'indirizzo di una struttura IntuiMessage), allora la precedente istruzione legge il dato dal campo im_Class di quella struttura IntuiMessage e lo copia nel registro D2.

Avremmo potuto, altrettanto facilmente, copiare il dato in qualche altra zona di memoria. Per esempio, per spostare dei dati in certe variabili chiamate *qualifier*, *code* e *class*, useremo l'istruzione MOVE:

```
move.w  im_Qualifier(a0), qualifier
move.w  im_Code(a0), code
move.w  im_Class(a0), class
```

Quali sono le differenze con la versione C dello stesso codice? Ebbene, se *message* è un puntatore alla struttura IntuiMessage, useremo gli operatori = e -> per ottenere lo stesso risultato:

```
qualifier = message->Qualifier;
     code = message->Code;
    class = message->Class;
```

Vediamo ancora dell'altro codice che dovrebbe esserci familiare: la combinazione Wait(), GetMsg(), leggi i dati e ReplyMsg() che serve a tenere sotto controllo il traffico di una porta IDCMP.

Prima la versione C:

```
Wait(bitmask);          /* aspetta fino a quando */
                        /* non succede qualcosa */
GetMsg(message);        /* leggi il messaggio */
class = message->Class;     /* leggi i dati */
code  = message->Code;
MouseX= message->MouseX;
MouseY= message->MouseY;

ReplyMsg(message);          /* rispondi al messaggio */
```

In assembly adottiamo esattamente lo stesso tipo di approccio, ma dobbiamo aderire alla convenzione d'uso dei registri, usata dalle routine di sistema.

La funzione *Wait* ha bisogno che gli venga fornito un bit-mask nel registro D0, mentre *GetMsg* e *ReplyMsg* hanno bisogno di trovare in A0 l'indirizzo della porta. Incidentalmente, CALLEXEC è la macro che serve a chiamare le routine della libreria Exec, presente nei file include di Amiga. Se mettiamo tutto quanto insieme, otteniamo il seguente codice assembly:

```
move.l    bitmask,d0
CALLEXEC Wait          ;aspetta fino a quando non
                       ;succede qualcosa
move.l    port,a0      ;indirizzo porta in A0

CALLEXEC GetMsg        ;prendi il puntatore al
                       ;messaggio
move.l    d0,a0        ;copialo in A0

move.l    im_Class(a0), class  ;leggi i dati
move.w    im_Code(a0), code
move.w    im_MouseX(a0), mouseX
move.w    im_MouseY(a0), mouseY

CALLEXEC ReplyMsg     ;rispondi al messaggio
```

Non penso che in termini di complessità ci sia una gran differenza fra le due versioni. Esiste, naturalmente, una zona nella quale noterete una differenza fra il C e l'assembly: il vostro portafoglio. Gli assembler per il 68000 sono parecchio più economici dei compilatori C.

Cambiamenti nel Disk Filing System

Cosa c'è veramente di nuovo nel Fast Filing System?

di Betty Clay

Siamo tutti a conoscenza dell'esistenza del nuovo Fast Filing System; appaiono ovunque prove e paragoni fra la sua velocità e quella del vecchio sistema nelle quali, fatta eccezione per il floppy disk, il nuovo sistema risulta l'incontrastato vincitore.

Per l'utente medio del filing system queste informazioni sono sufficienti. Funziona. È più veloce. 'Via il vecchio, sotto col nuovo'. Alcune persone arrivano addirittura a utilizzare il nuovo sistema anche su floppy, uso per il quale non era stato previsto, al quale si presta senza garantire sufficiente sicurezza e che, oltretutto, non risulta nemmeno più veloce dell'altro.

Nutrendo una grande curiosità nei confronti dei filing system, ho voluto vedere cosa c'era esattamente in questo FFS. Quando ci sono dei cambiamenti, voglio sapere esattamente in cosa consistono e in che maniera influenzano i miei dati. Se decido di leggere il settore di un disco con un disk editor, voglio sapere quali dati fanno parte delle informazioni di sistema e quali fanno parte del mio file.

Il fatto che ci sia un nuovo filing system implica che siano state apportate delle modifiche rispetto alle informazioni di sistema trattate negli articoli precedenti. Questo articolo si preoccupa principalmente di trovare queste modifiche, accompagnandone l'analisi con una quantità sufficiente di informazioni generali che lo rendono leggibile anche a coloro che non hanno speso molto tempo girovagando per i settori di un disco.

Non avendo un hard disk su cui provare il FFS e non volendolo adattare ai floppy, per i quali non è raccomandato, ho utilizzato il RAM disk recuperabile della Commodore, contenuto nel disco Workbench 1.3 (RAD:). Ho selezionato 80 cilindri nella Mountlist, quindi l'ho attivato usando il comando MOUNT e l'ho formattato con il FFS, usandolo per questo studio.

Che tipo di disco sono?

La prima cosa che il sistema legge da un disco, o dalla partizione di un hard disk, è il boot block. Questo contiene una long word che identifica il disco come disco DOS e, su floppy, potrebbe essere seguita dal programma necessario per fare il boot del sistema. Qui è disponibile una quantità di spazio considerevole: due settori da 512 byte ognuno, meno i 32 bit che identificano il tipo di DOS. Se il disco è stato appena formattato e installato, solo pochi di questi byte risultano utilizzati. Quelli rimanenti sono saltuariamente utilizzati per memorizzarvi delle protezioni da parte di molti sviluppatori software e per memorizzarvi diversi virus da parte di gente con la mente meno costruttiva.

Gli utilizzatori di hard disk sono stati, nella maggior parte dei casi, liberi dalle preoccupazioni riguardanti i virus del boot block, perché è il driver dell'hard disk che si preoccupa di effettuare il boot. Dell'intero boot block viene, quindi, usato solo l'identificatore di 32 bit. Se questo numero di identificazione viene alterato, il sistema provvederà a informarci che questo non è un disco DOS valido. Rimettere a posto questo identificatore è un buon punto di partenza nella serie di operazioni da effettuare per recuperare il contenuto di un disco.

Per quanto riguarda i boot block, l'unica differenza notata fra i due filing system consiste nell'ultimo byte della long word di identificazione: $444F5300 per il vecchio, $444F5301 per il nuovo. Questi numeri vengono tradotti nei corrispondenti caratteri *ASCII* DOS0 e DOS1 (dove 0 e 1 non sono caratteri ASCII, ma byte di quel valore; l'equivalente di CHR$(0) e CHR$(1)).

Quando il sistema legge questa long word, sa quale dei due filing system utilizzare; se non trova nessuno dei due in quella posizione, visualizza il requester 'Not a DOS disk'. In realtà, è stato possibile recuperare l'intero contenuto di alcune partizioni di hard disk, semplicemente mettendo a posto il valore di quella singola long word.

Ecco quello che ho trovato nei due boot block di un disco, formattato con il vecchio filing system, quando li ho esaminati con il programma *DiskED*. A questo punto devo dire che preferisco di gran lunga DiskED per questo tipo di studio, dal momento che individua chiaramente la posizione di ciascun oggetto, ma che comunque la Commodore avrebbe dovuto rifinirlo un po' e renderlo disponibile a un'audience più generale.

```
# sx                       ; visualizza in esadecimale
    00000000
```

```
# g 0              ; leggi il primo boot block
    Block 0 read (cyl 0, sur 0, sec 0)


# t 0 10           ; stampa prime 10 long word
    0:  444F5300
    1:  C0200F19
    2:  00000370
    3:  43FA0018
    4:  4EAEFFA0
    5:  4A80670A
    6:  20402068
    7:  00167000
    8:  4E7570FF
    9:  60FA646F
    10:  732E6C69
```

Ecco invece come si presenta l'FFS:

```
# sx               ; visualizza in esadecimale
    00000260


# g 0              ; leggi il primo boot block
    Block 0 read (cyl 0, sur 0, sec 0)


# t 0 10           ; stampa prime 10 long word
    0:  444F5301
    1:  444F5302   ; diversi perché RAD: non
    2:  444F5303   ; è stato 'installato'.
    3:  444F5304
    4:  444F5305
    5:  444F5306
    6:  444F5307
    7:  444F5308
    8:  444F5309
    9:  444F530A
    10:  444F530B
```

I tentativi di eseguire un INSTALL del RAD: sono andati a vuoto, quindi il boot block di questo esempio non contiene la sequenza di boot. La chiave, comunque, sta nel primo numero di ciascuna sequenza.

Alla radice di tutto

La directory root (radice) è la stessa, sia nel vecchio che nel nuovo sistema, o almeno così sembra a una prima occhiata. Beh, non è esattamente così.

Il blocco root è la chiave dell'intero disco in entrambi i sistemi. Esso è il settore centrale in ogni tipo di disco o partizione, rendendo così il tempo di seek (spostamento da una traccia all'altra) il più piccolo possibile per ogni settore. Gli altri blocchi del disco vengono raggiunti passando prima attraverso questo blocco e, se questo si danneggia, bisogna prendere misure drastiche per recuperarlo, oppure rinunciare al contenuto del disco.

La struttura *RootBlock*, così come le altre strutture che riguardano i dischi, è stampata in fondo a questo articolo.

Durante il boot, il sistema controlla il boot block e quindi la directory root. La sequenza di boot da disco dice al file system dove trovare la radice, quindi il driver di un hard disk dovrà fare altrettanto.

Nella directory root di entrambi i sistemi, i primi sei slot (ciascuno di 32 bit) contengono informazioni di sistema: il tipo di blocco (corto, settore singolo), due slot inutilizzati, le dimensioni della hash table (72 slot in un settore di 512 byte), uno slot riservato per usi futuri e uno contenente il checksum.

Questa parte è seguita da una tabella di settantadue slot (32 bit ognuno) che contiene i numeri dei settori nei quali troveremo le directory dell'utente o intestazioni di file (file header).

Il software di sistema è scritto in modo da poter continuare a funzionare se le dimensioni dei blocchi dovessero essere cambiate, ma nessun cambiamento in questo senso è stato finora annunciato. C'è, comunque, qualche indizio che fa pensare che qualche cambiamento possa essere fatto nella versione 1.4 del Kickstart, quindi i programmatori devono essere consapevoli del fatto che potremmo non avere per sempre questo standard di 512 byte per le dimensioni dei blocchi. Il software che si intende scrivere dovrebbe essere in grado di gestire anche altre dimensioni. La maggior parte del software non si deve preoccupare delle dimensioni dei settori di un disco, ma coloro che scrivono programmi per il recupero di dischi danneggiati o disk editor dovranno prevedere eventuali modifiche che potrebbero essere introdotte (e che probabilmente saranno introdotte) nelle future versioni del sistema operativo.

Benché la hash table non sia oggetto di particolare interesse nel corso di questo articolo, vale la pena notare che i valori di hash devono attualmente ricadere fra i valori 6 e 77 compresi. Il valore più basso è sei perché le informazioni di sistema occupano gli slot dallo zero al cinque.

L'algoritmo usato per calcolare il valore di hash è apparso ripetutamente in messaggi scritti da dipendenti Commodore:

```
Hash(name)
unsigned char *name;
    {
    int val,i;


    val = (int)*name++;
    for (i = 0; i < val; i++)
    val=((val*13)+(int)toupper(*name++))&0x7ff;
    return (val % 72);
    }
```

Ogni possibile nome di file uscirà da questo algoritmo con un valore compreso in questo intervallo. Il numero del blocco header di questo file verrà messo nello slot corrispondente a questo valore di hash. Nel caso di due file con lo stesso valore di hash, si ricorre a una catena di hash (hash chain).

Dopo la hash table c'è una long word che contiene i flag del bitmap, il cui valore dovrebbe essere -1 oppure 0.

Quando viene terminato l'aggiornamento del bitmap, questo flag viene messo a -1; se il sistema trova in questo posto uno 0 durante il boot, verrà invocato il disk-validator per ricostruire il bitmap. Questa operazione può richiedere un tempo considerevole e quindi quando accade ce ne accorgiamo. Ho letto di persone con hard disk che hanno dovuto aspettare dei minuti perché il validator finisse il suo lavoro e anche i floppy possono richiedere parecchio tempo. La maggior parte di noi, ormai, ha imparato ad aspettare che termini anche quell'ultima piccola operazione di scrittura, ma a volte diventiamo comunque impazienti e paghiamo caro per la nostra fretta la volta successiva che usiamo quel disco.

Nei due campi successivi troviamo ancora una differenza fra il vecchio e il nuovo sistema. Nel vecchio, i campi *BitMapKeys* e *BitMapExtension* (vedi le relative strutture in fondo all'articolo) sono invertiti e la *BitMapKeys* viene usata dalla posizione 24 in avanti.

Nel FFS, immediatamente dopo i flag di bitmap, troviamo una serie di 25 posizioni nelle quali memorizzare i numeri dei settori che contengono le bitmap. Ogni posizione in una bitmap può memorizzare l'informazione relativa a 32 settori, così, su floppy, un settore bastava sempre e tutte le altre posizioni erano vuote. Con gli hard disk il bisogno aumenta e il problema viene risolto grazie all'estensione dei bitmap.

Se l'ultima posizione nella tabella dei bitmap contiene un valore diverso da zero, il sistema capisce che ci sono altri blocchi bitmap elencati e quindi consulta il blocco *BitMapExtended* per trovarli. Quest'ultimo è uno solt da 32 bit contenente il numero del blocco che contiene i dati in eccesso.

Un blocco *BitMapExtension* non ha neppure il checksum per non occupare inutilmente spazio vitale. Contiene infatti 127 indirizzi di blocchi e il numero del prossimo blocco di estensione. Dal momento che tutte queste liste vengono lette dal fondo alla cima, il primo numero in questo settore è quello che punta al prossimo blocco di estensione.

Ecco degli estratti ottenuti da dischi OFS (vecchio) e FFS (nuovo). I dischi non sono identici, quindi alcune differenze nei numeri sono inevitabili e trascurabili.

```
# sd
    2

# t 75 83
    75:  0
    76:  1357          ; OFS Workbench
    77:  0
    78:  -1            ; il bitmap è valido
    79:  1029          : bitmap nel settore 1029
    80:  0
    81:  0
    82:  0
    83:  0
```

```
# g 880
    Block 880 read (cyl 40, sur 0, sec 0)

# t 75 83
    75:  0             ; FFS Workbench
    76:  0
    77:  0
    78:  -1            ; il bitmap è valido
    79:  881           ; bitmap nel settore 881
    80:  0
    81:  0
    82:  0
    83:  0
```

Una correzione a questa parte del blocco root ha permesso alla Commodore di rimuovere il limite di 53 Megabyte sulle dimensioni delle partizioni che affliggeva il vecchio file system.

Il vecchio sistema assumeva che il blocco root avesse i bit di protezione, come gli altri blocchi contenenti una directory, quindi procedeva a scrivere questi bit nel campo che puntava al blocco di estensione del bitmap. Questo succedeva perché i bit di protezione nelle directory d'utente occupano la stessa posizione usata per il numero del blocco di estensione nella directory root. Nel FFS questo bug è stato corretto, ma gli utenti del vecchio sistema continueranno ad avere questa limitazione di 53 Megabyte.

La parte rimanente della directory root è la stessa in entrambi i sistemi: la data dell'ultima modifica del blocco root, il nome del disco, la data dell'ultima modifica di un file o di una partizione (anche se quest'ultima parte non funziona ancora a dovere), la data di creazione del disco, alcuni slot riservati per usi futuri e infine una long word che identifica il tipo di blocco (numero secondario).

Nel FFS c'è ancora un errore nel Kickstart 1.3 che causa la scrittura della data dell'ultima modifica al blocco root anche nello slot riservato alla data dell'ultima modifica a un file o a una partizione.

I veri blocchi di bitmap, a differenza dei blocchi di estensione, *hanno* un checksum. Ricordate che i blocchi di estensione, invece, contengono solo altri numeri di settore.

I blocchi di bitmap indicano quali settori del disco sono usati e quali sono liberi. Un blocco bitmap ha il primo slot occupato da un checksum, seguito da 127 long word, ognuna rappresentante 32 settori, che fa 4064 settori per blocco. Questo numero non arriva a coprire 2 Mbyte, nel caso vogliate calcolare il numero di blocchi di bitmap necessari al vostro disco o alla vostra partizione. Se un bit di queste 127 long word è settato a uno, il blocco al quale corrisponde è disponibile all'uso. Quando questi è già in uso, invece, il bit viene messo a zero.

A prima vista questo metodo può sembrare l'incontrario di quanto converrebbe fare normalmente. Riflettendo, possiamo vedere che l'avere usato uno zero per indicare un blocco libero avrebbe richiesto una esame individuale di ogni bit, ma il test per verificare che una long word non sia a zero permette di

controllare 32 bit alla volta. Se si rileva un numero diverso da zero, allora quella long word può essere esplorata per vedere quali blocchi siano disponibili. Questo metodo risulta più veloce di quello di controllare ogni long word per trovare un bit a zero.

Per quanto ne so, non ci sono stati cambiamenti nella struttura di una directory utente sotto fast file system. I primi 78 slot sembrano essere gli stessi contenuti in una directory root.

Le directory d'utente non necessitano di informazioni relative al bitmap, quindi queste sono sostituite da posizioni inutilizzate (sette, ma non contigue), dai bit di protezione e da un campo di commento, 80 caratteri del quale sono a disposizione dell'utente. C'è solo un campo contenente la data in una directory utente, occupato dalla data di creazione della directory stessa.

Il campo riservato al nome della directory d'utente mette a disposizione 36 caratteri, anziché 40, ma questo è più che sufficiente, dal momento che il nome, in entrambi i casi, è limitato a 30. Una directory utente può essere parte di una catena, quindi c'è una posizione riservata al numero del prossimo blocco della catena, una per il numero di blocco della directory precedente e una per il numero secondario del tipo di file.

Il concatenamento degli hash è interessante. Prima di passare ad Amiga usavo un computer Commodore molto più semplice, il quale aveva una singola directory (nel caso dell'8050 due directory). Queste erano locate, come nel caso della nostra root, al centro del disco e ognuna conteneva le informazioni di tutti i file presenti sul disco. Era molto facile leggere le directory assai velocemente, ma era altrettanto facile esaurire lo spazio disponibile per aggiungere nuovi file. Ogni directory poteva contenere un numero finito di file e, una volta raggiunto questo numero, il disco risultava pieno, senza badare a quanto spazio rimanesse effettivamente libero sul supporto magnetico.

Nell'Amiga il concatenamento dei valori di hash ha risolto questo problema. Se due nomi di file ottengono lo stesso numero di hash dall'apposita routine, il primo viene messo nella directory root e il secondo viene raggiunto tramite la posizione riservata alla catena hash che si trova nell'header del primo file.

Pertanto, se il mio primo file è stato messo nello slot numero nove della directory root, quando salvo un file diverso con un hash uguale a nove, il sistema si sposta alla posizione nove della root, legge il settore lì indicato e paragona il nome di quel file con quello che ho indicato. Se questi sono differenti, viene letto il contenuto dello slot della catena hash e viene caricato il settore eventualmente indicato. Il processo si ripete fino a quando non si trova il file desiderato. L'ultimo file della catena ha uno zero in questa posizione, così il file system sa quando interrompere la ricerca.

Allora cosa c'è veramente di nuovo?

I cambiamenti nei blocchi boot, directory e bitmap sono piccoli. È quando arriviamo ai blocchi data che i cambiamenti nel FFS diventano veramente visibili.

Una veloce occhiata a uno qualsiasi degli altri blocchi non mostrerebbe alcuna differenza, anche se ce ne sono alcune, che abbiamo visto sopra. Anche un file header non mostra alcuna differenza di rilievo. Secondo un messaggio apparso su *Usenet*, il numero di blocchi del file (terza posizione nel blocco header) non viene aggiornato dal fast filing system, ma rimane tuttavia presente nella sua posizione.

Il prossimo elemento nella catena per arrivare a un file è il blocco file header, l'intestazione di un file, che serve a vari scopi. Innanzitutto, questo tipo di blocchi viene utilizzato nelle strutture *FileLock*. La Commodore ci ha detto che le strutture *FileLock* continueranno a essere supportate in futuro.

Quando un file viene bloccato (locked), il campo *fl_*key di *FileLock* conterrà il numero del blocco header del file. Questo blocco individua l'inizio del file in un disco o in una partizione, o, nel caso di un RAM disk, l'indirizzo al quale il file è attualmente memorizzato.

Il blocco header contiene le informazioni riguardanti il file a cui appartiene: il suo nome, quando è stato creato, i suoi bit di protezione e i puntatori ai blocchi che contengono i dati e a quelli di estensione.

I blocchi file header usano le prime sei posizioni per informazioni di sistema come il tipo di blocco, il numero del blocco, il numero di blocchi nel file, il numero del primo blocco dati e il checksum.

Queste informazioni sono seguite da una tabella con 72 posizioni, ognuna delle quali contiene il numero di un blocco dati usato in questo file. Gli slot vengono riempiti a partire dal numero 78 per finire al numero 6. Se il file dovesse richiedere più di settantadue settori, il numero del blocco di estensione verrà posizionato nello slot 126.

Dopo la tabella si trovano le solite posizioni riservate per usi futuri (tre in questo caso) e quelle necessarie a immagazzinare la data e il nome del file. Segue lo slot della catena hash, il numero della directory genitore, il numero del blocco di estensione e il numero secondario del tipo di file (-3 in questo caso).

Quando arriviamo ai veri blocchi dati, comunque, troviamo un cambiamento veramente molto grosso. Le prime sei long word nel vecchio sistema contenevano informazioni di sistema molto simili a quelle che si trovano nelle directory. Questo lasciava spazio per soli 488 byte di dati, dal momento che gli altri 24 erano utilizzati in questa maniera. Il fatto non occupava solo uno spazio maggiore del disco, ma rallentava anche il file system.

Proviamo a cercare un file, utilizzando DiskED, in entrambi i file system e quindi facciamo un paragone. Il valore hash di *c* è 14; quello di *run* è 6. Cominciamo quindi dal principio: troviamo la directory *c* e rintracciamo il comando *run* nei due modi:

Per trovare la directory c, prima leggiamo il blocco root:

```
FastFileSystem          OldFileSystem
```

```
  0:  2               0:  2       ;tipo di
                                  ;blocco (corto)
  1:  0               1:  0
  2:  0               2:  0
  3:  72              3:  72      ;dimensioni
          ;hash table
  4:  0               4:  0
  5:  -1199734386     5:  9069645323 ;checksum
  6:  0               6:  0
  7:  0               7:  882
  8:  0               8:  0
  9:  0               9:  0
  10:  0              10:  884
  11:  0              11:  0
  12:  0              12:  0
  13:  0              13:  885
  14:  882            14:  887    ;blocco
                                  ;contenente la
                                  ;directory C
```

Successivamente, leggiamo il settore che contiene la directory C ed esaminiamo l'elemento nello slot 6, che è il valore hash di *run*.

```
  FastFileSystem              OldFileSystem

  # g 882                     # g 887       ;leggiamo il
                                            ;blocco dove
                                            ;si trova C
  Block 882 read              Block 887 read
  (cyl 40, sur 0, sec 2)      (cyl 40, sur 0, sec 7)

  # t 0 10                    # t 0 10      ;stampa le
                                            ;prime 10
                                            ;posizioni
  0:  2                       0:  2         ;tipo di blocco
                                            ;(corto)
  1:  882                     1:  887       ;numero di
                                            ;questo blocco
  2:  0                       2:  0
  3:  0                       3:  0
  4:  0                       4:  0
  5:  -23315566               5:  -23327944 ; checksum
  6:  883                     6:  888       ;blocco header
                                            ;di 'run'
  7:  885                     7:  890
  8:  889                     8:  897
  9:  891                     9:  899
  10:  895                    10:  901
```

I blocchi header sono gli stessi per i due file system. È necessario visitare questo blocco, comunque, per trovare i numeri dei settori che contengono i dati reali di 'run'. Adesso ci spostiamo al blocco 883 sul FFS e al blocco 888 sul OFS, i quali contengono il blocco file header di *run*:

```
  FastFileSystem          OldFileSystem
  0:  000003F3            6:  000003F3    ;hunk header,
  0:  2                   0:  2           ;Tipo di blocco
                                          ;(short)
```

```
  1:  883            1:  888      ;numero di
                                  ;questo blocco
  2:  6              2:  6        ;settori usati in
                                  ;questo file
  3:  0              3:  0        ;data block per
                                  ;questo file
  4:  884            4:  889      ;primo blocco
                                  ;del file
  5:  -55752819      5:  -55752929
  6:  0              6:  0
  7:  0              6:  0
  ..................................
  71:  0             71:  0
  72:  906           72:  873     ;ultimo blocco
                                  ;usato da 'run'
  73:  905           73:  872
  74:  904           74:  871
  75:  903           75:  870
  76:  902           76:  879     ;secondo blocco
                                  ;usato da 'run'
  77:  884           77:  889     ;primo blocco
                                  ;usato da 'run'
  78:  0             78:  0
```

E, quando leggiamo il blocco 884 del FFS, oppure il blocco 887 del OFS, osserviamo infine la maggiore differenza tra i due sistemi. Qui, ho cambiato dalla notazione decimale a quella esadecimale, per mostrare con maggior chiarezza che cosa succede esattamente.

```
  FastFileSystem          OldFileSystem
                          # t 0 20
     0:  00000008    ;tipo di
                                  ;blocco (data)
     1:  00000378    ;numero di
                                  ;questo settore
     2:  00000001    ;posizione
                                  ;nel file
     3:  000001E8    ;ammontare dei
                                  ;dati (488 byte)
     4:  0000036F    ;prossimo blocco
                                  ;nel file (879)
     5:  7CDCFCB7    ;Checksum
  0:  000003F3            6:  000003F3   ;hunk header,
                                         ;inizio del file
  1:  00000000            7:  00000000
  2:  00000002            8:  00000002
  3:  00000000            9:  00000000
  4:  00000001            10:  00000001
  5:  0000004F            11:  0000004F
  6:  00000226            12:  00000226
  7:  000003E9            13:  000003E9  ;hunk_code
  8:  0000004F            14:  0000004F
  9:  286A0164            15:  286A0164
  10:  700C4E95           16:  700C4E95
```

Come potete vedere, i dati sono identici, ma il vecchio file system ha 24 byte di informazione che non viene utilizzata nel fast system.

Questo velocizza le cose in molti modi. Ovviamente ci sono più dati in ogni settore, quindi si effettuano meno operazioni di lettura. Meno ovviamente, bisogna compiere alcune azioni per rendere i dati pronti all'uso.

Sotto OFS la traccia viene letta nell'apposito buffer e decodificata, quindi i primi 24 byte all'inizio di ogni settore sono rimossi prima che l'informazione venga copiata nel buffer dell'utente. Con il nuovo sistema tutta l'operazione può essere omessa e i dati vengono trasferiti tramite DMA nel buffer dell'utente. Questo risparmio, moltiplicato per ogni settore, può risultare in grosse differenze nella quantità di tempo richiesto per portare a termine l'intera operazione.

Per ottenere questo risparmio di spazio e di tempo sono stati però sacrificati alcuni mezzi per il recupero di file danneggiati. A prima vista può sembrare che nel FFS non ci sia alcun mezzo per poter recuperare alcunché, ma questo non è completamente vero.

In aggiunta ai blocchi che vediamo, esistono zone di intestazione (label area) per ciascuno di questi settori. Non ho trovato documentazione per queste zone sotto FFS, ma sotto il vecchio sistema, il RKM (Rom Kernel Manual) ci diceva che il loro contenuto è:

```
2 byte a 00
2 byte a A1 (byte di sincronismo MFM)
1 byte per indicare il tipo di formato
1 byte per il numero della traccia
1 byte per il numero del settore
16 byte non utilizzati sotto OFS
   (credo che non vengano utilizzati neanche sotto
   FFS),presenti per usi futuri per il recupero
   degli errori
4 byte per il checksum di questa zona
4 byte per il checksum del blocco dati
```

Le informazioni relative alla traccia, al settore e al checksum sono ancora disponibili, qui nella label del settore che non vediamo nei nostri disk editor. Era ridondante ripetere queste informazioni nelle prime sei posizioni del settore, anche se si è rivelato utile per recuperare dischi e per salvaguardare l'integrità dei file.

I blocchi dati non hanno più il puntatore al loro header genitore e adesso dipendiamo interamente dai questi ultimi per recuperare i dati.

Cosa riserva il futuro?

Le versioni future del sistema operativo porteranno cambiamenti assai più signiicativi di quelli arrivati con l'1.3. Uno studio sull'organizzazione del sistema ci ha detto fin dal principio che non avremmo sempre avuto blocchi da 512 byte l'uno ed è abbastanza probabile che cominceremo ad avere blocchi di dimensioni variabili già con la versione 1.4.

Le maggiori dimensioni dei blocchi comporteranno una diminuzione delle operazioni di lettura e delle operazioni di spostamento delle testine, rendendo più veloce il sistema. Le tabelle hash saranno finalmente più lunghe di 72 slot, favorendo una riduzione del numero di collisioni dei valori di hash e riducendo il ricorso al loro concatenamento. Le fondamenta per entrambe queste migliorie sono già presenti, quindi possono essere implementate con inconvenienti ridotti al minimo. Probabilmente seguiranno altri cambiamenti.

Il software che accede al disco, seguendo le procedure raccomandate ufficialmente, continuerà a funzionare (obbedisci alle regole e tu e il tuo software soppravviverete).

Colui che utilizzerà metodi scorretti per guadagnare un temporaneo vantaggio in velocità, soccomberà quando il sistema verrà cambiato.

Strutture C per i diversi blocchi del disco (basati su quelle pubblicate in AmigaMail Novembre/Dicembre 1988):

Blocco root

```
struct RootBlock {
LONG   Type;             /* corto (2) */
ULONG  OwnKey;           /* non utilizzato, 0 */
ULONG  SeqNum;           /* non utilizzato */
ULONG  HtSize;           /* numero di slot nella hash table */
ULONG  Nothing1;         /* uso riservato, messo a zero */
LONG   Checksum;
ULONG  HashTable[72];    /* hash table con 72 slot */
LONG   BitmapFlags;      /* -1 se valida; 0 se non validata */
ULONG  BitmapKeys[25];   /* per FFS; OFS ha qui BitmapExtended */
```

```
ULONG   BitmapExtend;       /* dov'è la prossima lista di BitmapKeys? */
                            /* Solo nel FFS. L'OFS aveva un problema con */
                            /* questo slot, come spiegato nell'articolo */
ULONG   DirAltered[3];      /* data dell'ultima modifica effettuata /*
char    Name[40];           /* stringa BCPL, nome del disco */
ULONG   DiskAltered[3];     /* attualmente c'è un bug nel FFS, dovrebbe */
                            /* mostrare la data dell'ultima alterazione */
                            /* di un file su questo disco */
ULONG   DiskMade[3];        /* data di formattazione del disco */
ULONG   Nothing2;           /* uso riservato */
ULONG   Nothing3;           /* uso riservato */
ULONG   Nothing4;           /* uso riservato */
LONG    SecondaryType;      /* tipo 1, ROOT */
}
```

Directory

```
struct UserDirectoryBlock {
LONG    Type;               /* corto (2) */
ULONG   OwnKey;             /* numero di questo settore */
ULONG   Spare1;             /* tutti i campi 'spare' sono messi a zero */
ULONG   Spare2;             /* e riservati per un uso futuro */
ULONG   Spare3;
LONG    Checksum;
ULONG   HashTable[72];      /* numero di slot disponibili in questa tabella */
LONG    Spare4;
LONG    Spare5;
ULONG   Protection;         /* bit di protezione per questa directory */
LONG    Spare6;
char    Comment[92];        /* solo 80 disponibili all'utente */
ULONG   Created[3];         /* data di creazione o modifica della directory */
char    DirName[36];        /* nome della directory, solo 30 byte usati */
LONG    Spare7[7];
ULONG   HashChain;          /* dov'è il prossimo file con lo stesso valore */
                            /* di hash? (0 se non ce ne sono più) */
ULONG   Parent;             /* blocco della directory genitore */
ULONG   Spare8;
LONG    SecondaryType;      /* tipo 2, ST_USERDIR */
}
```

File header

```
struct FileHeaderBlock {
LONG    Type;               /* corto (2) */
ULONG   OwnKey;             /* numero di questo settore */
ULONG   HighSeq;            /* quanti blocchi sono utilizzati per questo */
                            /* file? Usato nel OFS, non aggiornato nel FFS */
ULONG   DataSize;           /* numero di blocchi dati in questo file */
ULONG   FirstBlock;         /* da che settore comincia questo file? */
LONG    Checksum;
ULONG   DataBlocks[72];     /* elenco fino a 72 settori usati in questo */
                            /* file, cominciando dal #71, andando verso */
                            /* il #0. Se ne occorrono di più si usa il */
                            /* blocco di estensione */
ULONG   Spare1;
ULONG   Spare2;
```

```
ULONG  Protect;           /* bit di protezione per questo file */
ULONG  Date[3];           /* data di creazione del file */
char   FileName[36];      /* nome del file, solo 30 sono usati */
ULONG  Spare3[7];
ULONG  HashChain;         /* blocco header del prossimo file che */
                          /* ha lo stesso valore hash di questo */
ULONG  Parent;            /* blocco della directory genitore */
ULONG  Extension;         /* puntatore a un altro blocco come questo */
                          /* che contiene un'altra parte della lista */
                          /* dei blocchi dati di questo file */
LONG   Second;            /* tipo -3 */
}
```

I blocchi dati sotto FFS non hanno alcuna struttura e sono interamente riempiti di dati. Per il OFS la struttura dei blocchi dati è la seguente:

```
struct FileDataBlock {
LONG   Type;              /* tipo 8, blocco dati */
ULONG  OwnKey;            /* numero di questo settore */
ULONG  SeqNumber;         /* numero di sequenza di questo blocco nel file */
ULONG  DataSize;          /* quanti byte di dati in questo blocco? */
ULONG  NextBlock;         /* quale settore ospita prossimo blocco dati? */
LONG   Checksum;
char   Data[488];         /* 488 byte di dati */
}
```

(segue da pagina 25)

Linguaggio Assembly

Possiamo adesso chiedere all'assembler di mettersi al lavoro. La maniera normale consiste nel digitare <*nome dell'assembler*> TEST. Controllate nella sua documentazione, nel caso sia richiesto qualcosa di diverso.

Se ci vengono segnalati degli errori, torniamo indietro, correggiamoli e riproviamo ad assemblare. Alla fine il programma effettuerà un passaggio senza segnalare problemi e produrrà un file *oggetto*, riconoscibile dal suffisso .o oppure .obj. In altre parole, dovremmo avere adesso un file chiamato TEST.O o TEST.OBJ. Ma il nostro programma non è ancora pronto per essere eseguito.

Alcuni assembler, come AssemPro e Devpac, saltano la fase di link e generano direttamente un eseguibile; se questo è il vostro caso, non avrete bisogno del linker.

```
Il comando     BLINK TEST.O TO TEST
oppure         BLINK TEST.OBJ TO TEST
```

convertirà il nostro programma in forma eseguibile, pronto per essere lanciato. Digitiamo TEST e il computer dovrebbe rispondere *Hello*, World!.

Avete appena messo assieme il vostro primo semplice programma in linguaggio assembly per Amiga. Come vi sentite?

Devpac Amiga della HiSoft

Una recensione della versione 2 di questo famoso pacchetto di sviluppo per 68000

di Danny Ross

Danny Ross sta frequentando l'ultimo anno di corso di Scienze dell'Informazione. Il suo tempo libero si divide fra lo sviluppo di algoritmi di incanalamento per network di transputer, fra la programmazione di Amiga come freelance e fra la scrittura di software per il PC. Vivendo in una casetta in stile Gotico nel Galles del sud, con altri due maniaci della programmazione, la sua idea di rilassamento consiste nell'esplorare tutte le ultime chiamate delle librerie di Amiga!

Chiunque abbia usato un prodotto della HiSoft si troverà immediatamente a suo agio con *Devpac* Amiga; tutto è integrato in quel familiare ambiente di sviluppo che è diventato una caratteristica dei linguaggi della HiSoft.

Con la versione 2 di Devpac, la HiSoft ha nettamente migliorato il loro già eccellente sistema di sviluppo per 68000, rendendolo indispensabile per il programmatore serio.

GenAm2 - l'editor/coordinatore

Devpac Amiga è scomposto in tre programmi principali, GenAm2, GenIm2 e MonAm2. Digitando *GenAm2* nel CLI (non esiste, stranamente, alcun supporto per il Workbench) invocheremo la finestra dell'editor GenAm2 contenente una breve linea di stato, in fondo, che riporta la posizione del cursore e la memoria disponibile.

L'editor è configurato inizialmente con un buffer per il testo di 60000 byte, che può essere aumentato o diminuito per mezzo di una opzione nel menu Preferences. Svariate altre caratteristiche, come la dimensione dei tab, l'autoindentazione e il ritorno automatico, sono ora disponibili in questo menu e possono essere modificati con il mouse direttamente all'interno dell'editor (le versioni precedenti richiedevano un programma separato, GenIns).

I comandi dell'editor non sono cambiati molto e le orribili e antiquate combinazioni di tasti usate dal WordStar sono tuttora supportate. Fortunatamente, adesso le operazioni di selezione, taglio e inserimento di blocchi mettono in evidenza il testo su cui si svolge l'operazione, pur utilizzando ancora i tasti di funzione. Personalmente avrei preferito lo standard utilizzato nei programmi TxED/CygnusED.

Una volta inserito il nostro codice sorgente, GenAm2 comincia veramente a trovare il suo ritmo. Selezionando *Assemble* dal menu *Program* (o premendo Amiga-A) verrà visualizzato una finestra di opzioni estremamente utile (novità nel 2.0). Alcune di queste opzioni sono semplicemente dei sostituti per le direttive vecchio-stile dell'assembler (OPT L+ e il resto, che sono ancora disponibili), ma altre sono completamente nuove. Adesso è possibile dire a GenAm2 di mettere il codice prodotto nel disco o nella memoria.

GenIm2 - l'assembler

Quando sono state effettuate tutte le selezioni del caso, si seleziona il gadget *Assemble* nella finestra delle opzioni di GenAm2, che, oltre a essere un editor, funge anche da coordinatore per gli altri due programmi del pacchetto. Viene quindi caricato l'assembler vero e proprio, chiamato GenIm2, che procede alla elaborazione del codice sorgente. Dopo essere stato chiamato per la prima volta, GenIm2 rimane in memoria, pronto a essere riutilizzato per la prossima volta, eliminando così i tempi morti di caricamento.

Con il codice appena prodotto in memoria, il debugger MonAm2 può essere invocato da GenAm2, permettendo di ottenere un ciclo di edit-assemble-debug molto rapido. Altre opzioni che entrano in gioco durante l'assemblamento di un file comprendono una direttiva 'Fast', che velocizza l'assemblaggio trattenendo i file *include* in memoria (richiede abbastanza RAM libera), uno switch per ottenere codice eseguibile o linkabile e la possibilità di specificare la destinazione del file .lst (listato).

Queste e altre opzioni possono essere selezionate nell'apposita finestra di opzioni di GenAm2, oppure possono essere specificate sulla linea di comando quando si lancia GenIm2 da CLI. L'assembler può essere, infatti, utilizzato sia nell'ambiente integrato GenAm2-GenIm2-MonAm2, sia come programma a sé stante, come i più classici assembler disponibili sul mercato. In

questo modo, chi preferisce, può continuare a usare il suo editor preferito per lavorare sul sorgente, quindi lanciare GenIm2 per assemblare e poi utilizzare MonAm2 o un altro debugger di propria scelta.

Adesso è supportata l'intera serie di macro caratteristiche degli assembler Amiga, quindi GenIm2 può rimpiazzare prodotti come quello della Metacomco senza problemi.

MonAm2 - il debugger simbolico

La terza parte di Devpac è costituita dal programma *MonAm2*. Questo è il versatile debugger simbolico della HiSoft che può lavorare come programma di utilità a sé stante oppure in coppia con GenAm2. Quando GenAm2 è attivo, MonAm2 è disponibile dal menu *Program* per lavorare sia sul codice prodotto da MonIm2 in memoria, sia su qualsiasi file eseguibile AmigaDOS.

La prima cosa che si nota di MonAm2 è che lavora su un proprio schermo. Questa è un'idea eccellente perché significa che il programma sotto esame può aprire i propri screen e definire qualsiasi tipo di display, senza influenzare le sessioni di debugging di MonAm2 (assumendo che il programma in questione usi delle chiamate legali).

Naturalmente, se apriamo uno schermo senza barra di spostamento (drag bar) o senza gadget di profondità, allora potremmo accorgerci di avere perso lo schermo di MonAm2, nonostante ci sia una funzione incorporata in MonAm2 che manda lo schermo sullo sfondo. Magari la HiSoft potrebbe fornire un programma di utility che permetta di passare da uno schermo all'altro mediante l'uso di una hot-key. Utility di questo genere si possono comunque trovare nel vasto oceano di software di pubblico dominio.

Come menzionato in precedenza, MonAm2 è un debugger simbolico. Operando le appropriate selezioni all'interno di GenAm2, questi produrrà un file di output che contiene informazioni sui simboli presenti nel programma. Caricando questo programma con MonAm2, tutte le etichette che abbiamo usato nel nostro sorgente verranno visualizzate nel codice disassemblato, cosa che può rendere l'operazione di debugging più facile di svariati ordini di grandezza.

MonAm2 è composto di 3 finestre, con una quarta opzionale che può essere usata per visualizzare un file di testo (normalmente il nostro codice sorgente). Ogni finestra mostra informazioni utili riguardanti il programma in esecuzione. La più grande delle tre contiene la stampa di tutti i registri del 68000 (inclusi SR, SSP e il PC).

La seconda in ordine di grandezza è la finestra Disassembly, che normalmente visualizza il codice negli immediati paraggi del Program Counter. Parte di questa finestra deve essere sacrificata se si utilizza la possibilità di vedere il file sorgente nella sua finestra separata. La terza finestra è una finestra diretta sulla memoria.

Ci sono, in effetti, altre due finestre che rimangono quasi sempre nascoste alla vista. Una contiene delle informazioni di help, che in realtà non danno un grande aiuto, ma perlomeno fornisce informazioni dettagliatissime sulla sitazione attuale dell'occupazione di memoria. L'altra è una finestra che tiene una traccia delle ultime istruzioni eseguite e dello stato dei registri durante la loro esecuzione. Quest'ultima finestra è molto preziosa e fa di MonAm2 un prodotto molto più utilizzabile.

Tutti i comandi di MonAm2 sono impartiti da tastiera. Sebbene questo consenta la massima rapidità d'uso, è molto scomodo *dover* ricordare le varie combinazioni di tasti normali oppure associati a CTRL o ad AMIGA (non che alcuni di questi non siano perfettamente azzeccati, come la combinazione che attiva *Trace Call* e *Set Breakpoint After Current Instruction*.

Naturalmente, il mancato utilizzo dell'interfaccia grafica permette a MonAm2 di rimanere contenuto come dimensioni e l'ultima cosa che vorremmo, durante il debugging di un grosso programma, sarebbe di avere un debugger di 200K piazzato in sottofondo. Nonostante questa considerazione, MonAm2 potrebbe fare certamente un uso migliore di Intuition, perlomeno con un box per la selezione dei file!

Conclusione

L'intero processo di edit-assemble-debug è stato velocizzato in maniera significativa, facendo di GenAm2 un prodotto straordinariamente brillante. In paragone, *AssemPro* è una schifezza.

Pur formando, sotto il controllo dell'editor GenAm2, un ambiente perfettamente integrato, l'assembler GenIm2 e il debugger MonAm2 possono essere utilizzati in maniera completamente indipendente, adattandosi a qualsiasi ambiente di lavoro già utilizzato dal programmatore.

Il pacchetto Devpac della HiSoft, pur beneficiando di dimensioni contenute, è molto potente per lo sviluppo sul microprocessore 68000. Ci sarebbero moltissime opzioni e funzioni che potrebbero essere aggiunte, ma al costo di un aumento delle dimensioni che risulterebbe inaccettabile per la maggior parte degli sviluppatori (MonAm2 è lungo 25788 byte, GenIm2 29004 byte e GenAm2 appena 23744 byte).

Sebbene la HiSoft dica che Devpac è adatto per l'intera famiglia 680x0 (implicando, quindi, il supporto perlomeno del 68010), entrambi GenAm2 e MonAm2 non sono stati in grado di compilare o disassemblare correttamente il mio codice per 68010. La migliore performance è venuta da GenAm2 che mi ha detto che *MOVE* CCR,D0 era un'istruzione per 68010 e che l'aveva sostituita con *MOVE* SR,D0.

Con un prezzo molto ragionevole di 59.95 sterline inglesi (o meno), gli utenti di precedenti versioni di Devpac dovrebbero certamente pensare a passare alla nuova versione e chiunque desideri un ambiente di sviluppo integrato e veloce su 68000 dovrebbe prendere in seria considerazione l'acquisto di questo prodotto.

HiSoft, The Old School, Greenfield, Bedford, MK45 5DE, Inghilterra. Tel: 0044-525-718181

Lattice C 5.0

Un'analisi del nuovo sistema di sviluppo Lattice C

di Peter Booth

È disponibile ormai da un po' di tempo la versione 5.0 del compilatore C della Lattice e ci sembra che sia dunque opportuno descrivere le caratteristiche di questo nuovo prodotto. Questa versione si distingue dalle precedenti per il miglioramento della adesione allo standard ANSI e per un certo numero di miglioramenti e aggiunte.

I manuali

Il sistema di sviluppo offerto dalla Lattice comprende cinque dischi e due manuali in formato A5 di buona fattura. La User Guide fornisce un'introduzione ragionevolmente graduale per i nuovi utenti, precisa le differenze tra il Lattice C e quello definito nel K & R e presenta le caratteristiche generali dell'ambiente di programmazione C su Amiga. Il resto del materiale è sostanzialmente costituito da manuali di riferimento relativi ai vari moduli, comandi e programmi di utilità.

La sezione che tratta le routine di libreria, il cui numero è costantemente in aumento, è stata molto curata, soprattutto per quanto riguarda la presentazione e gli indici.

Molta dell'informazione presente nei manuali viene fornita allo scopo di dare una descrizione il più esauriente e completa possibile per cui può capitare, come, del resto, spesso accade per quei pacchetti software che sono rivolti al mercato professionale, che i nuovi utenti e i principianti trovino i manuali di ardua lettura e comprensione. Fortunatamente nel manuale e sui dischetti sono riportati alcuni semplici esempi che possono aiutare il principiante nell'uso del compilatore.

Due compilatori e...

Nel package possiamo trovare due versioni del compilatore, il linker Blink, il Lattice Screen Editor, un ottimizzatore globale, un macro assembler, un profiler, una utility per l'analisi post-mortem dell'esecuzione di un programma, codice di supporto per i programmi del tipo load-and-stay-resident, un compressore di file ... e potremmo andare avanti ancora per molto.

Viene anche fornito l'intero insieme delle utility chiamato Compiler Companion, prima venduto separatamente, e il debugger a livello sorgente CodeProbe costituisce ora parte integrante del pacchetto di sviluppo.

La libreria delle funzioni è cresciuta e ora conta almeno 300 routine. Una gran parte delle funzioni della library è stata ricompilata con i nuovi tool e questo, in aggiunta all'uso di algoritmi migliorati e dell'assembler, ha comportato un ulteriore aumento delle prestazioni del codice ottenuto con questa nuova versione.

Come è logico aspettarsi, la versione 5.0 risulta completamente compatibile verso l'alto con la versione 4.0. Moduli oggetto e librerie possono essere liberamente mischiati, indipendentemente dal fatto che siano stati creati con l'una o l'altra versione; naturalmente, però, solo il codice compilato con la versione più recente può sfruttare appieno le potenzialità del nuovo compilatore, come il debugging a livello di sorgente.

Il compilatore e le librerie possono poi ottimizzare la generazione di codice, tenendo conto della eventuale presenza di un 68010, di un 68020 o di un 68030, come dei coprocessori 68881 e 68882.

Il compilatore, inoltre, ha ora un preprocessore che si conforma strettamente alle specifiche ANSI; viene riconosciuta anche la direttiva defined() e, in aggiunta, i simboli _DATE_ e _TIME_ forniscono la data e l'ora della compilazione. La Lattice ha dei propri rappresentanti nel comitato ANSI e quindi è in una eccellente posizione per tenersi aggiornata sulle eventuali nuove proposte. Questo fatto può far capire perché la Lattice abbia profuso un certo impegno per mantenere il proprio compilatore conforme allo standard ANSI e abbia previsto un notevole numero di diagnostici per individuare programmi non conformi allo standard.

Nel package sono presenti due differenti versioni del compilatore. La versione completa, che risulta solo 20K più grande, ha in più rispetto alla minore solo alcune possibilità relative all'elaborazione del listato: è possibile, infatti, ottenere la visione dell'espansione delle macro, del conteggio del livello di innestamento dei cicli e del contenuto dei file include. Inoltre, consente la generazione di file prototipo per tutte le funzioni che si trovano in un modulo. Quest'ultima possibilità elimina il compito potenzialmente noiosissimo della costruzione di una lista

dei prototipi di tutte le funzioni presenti in un progetto.

In entrambe le versioni, la gestione degli errori è stata sostanzialmente migliorata e, tra le altre cose, è ora possibile controllare il comportamento del compilatore in risposta a svariate condizioni di errore.

Per quanto riguarda le librerie, oltre ad amiga.lib e alle consuete librerie C Lattice, l'insieme (in continua crescita) delle varie librerie comprende anche un certo numero di librerie matematiche IEEE, FFP e per sfruttare la presenza dei coprocessori 68881 o 68882.

La chiamata diretta delle funzioni di library di Amiga può produrre significativi aumenti di velocità di esecuzione in qualsiasi programma. Nel corso dell'evoluzione del proprio compilatore, la Lattice ha tentato di migliorare le prestazioni evitando che le chiamate alle funzioni di sistema dovessero passare attraverso le routine di "interfacciamento" contenute in amiga.lib. Questo accorgimento permette di evitare il trasporto degli argomenti sullo stack (argomenti che, poi, venivano comunque portati nuovamente nei registri appropriati dalle routine di interfacciamento prima della chiamata vera e propria alla funzione). È ora possibile specificare al compilatore in quali registri vadano posti gli argomenti, lasciando quindi la possibilità al compilatore stesso di scegliere il metodo ottimale di caricamento dei parametri nei registri. L'uso diretto delle funzioni di sistema è ottenuto grazie all'impiego di speciali file header presenti nel package.

Funzioni e parole riservate

La versione attuale del compilatore comprende diverse funzioni speciali "built-in", scritte con lo scopo di generare codice 680x0 di alta qualità. L'uso di funzioni di questo è perfino incoraggiato dallo stesso standard ANSI. Le funzioni speciali sono abs, memset, putreg, strlen, max, memcmp, getreg, strcmp, min, memcpy, emit, strcpy e geta4. Il compilatore riconoscerà automaticamente una funzione "built-in" qualora il suo identificatore sia preceduto da __builtin_.

Vengono ora riconosciute diverse parole riservate, comprese le keyword a norme ANSI const, enum, void e volatile. Le parole riservate chip, far e near sono estensioni allo standard ANSI e dovrebbero essere precedute da due trattini; il compilatore Lattice, comunque, riconosce ambedue le forme. La più importante tra le parole riservate relative alla classe di memoria è la keyword chip (o meglio __chip), che permette di obbligare il compilatore a mettere l'oggetto a cui si riferisce nella memoria CHIP. In altri termini, ciò consente di scrivere definizioni del tipo:

```
static  USHORT chip GadgetImageData[] = {
0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,
0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,
0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,
0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,0x7fff,
... e così via ...
};
```

Inoltre, sono previste altre parole riservate da applicarsi a funzioni per specificare particolari convenzioni di passaggio dei parametri. Di una certa importanza sono __regargs e __stargs che indicano se si intende seguire una convenzione di passaggio dei parametri basata sui registri o basata sullo stack. Viene anche messa a disposizione del programmatore tramite la parola riservata __asm la possibilità di specificare in quale registro passare un certo parametro (la maggior parte dei programmatori che hanno bisogno di una simile possibilità non esiteranno ad usare l'assembler). Personalmente mi domando se una simile possibilità sia realmente necessaria o addirittura auspicabile, ma può essere che non mi renda esattamente conto dei problemi che hanno giustificato la sua inclusione nel compilatore.

Le utility "Compiler Companion"

Non è passato molto tempo da quando la Lattice ha annunciato la disponibilità di un insieme di utility, che assistono il programmatore nello sviluppo di programmi, raccolte sotto il nome di "Compiler Companion". Alcune di queste sono state immesse sul mercato da parecchio tempo, ma dal momento che rimangono comunque utili, la Lattice ha deciso di includerle nel package di sviluppo.

BUILD permette l'inserimento o l'alternanza di linee di testo in un dato file.

CXREF genera una cross-reference da un file sorgente C. Produce tabelle in cui sono riportate definizioni, funzioni, etichette e così via e fornisce l'informazione necessaria per determinare dove esse compaiano nel programma.

DIFF è una utility che consente di confrontare i file e di determinare le differenze tra essi.

EXTRACT, come già il nome suggerisce, estrae i nomi di file da una directory. Questo comando viene spesso usato in associazione a BUILD per creare file batch per consentire di eseguire automaticamente lunghe sequenze di comandi.

FILES è un tool molto potente che permette di copiare, cancellare o fare ricerche su file o intere directory. Può trattare anche directory innestate una dentro l'altra e può limitare le operazioni di ricerca solo ai quei file che hanno una certa data di creazione o certe dimensioni.

GREP sta per Global Regular Expression search and Print ovvero utility di impiego generale per la ricerca e la stampa di espressioni regolari. Essa serve per ricercare sequenze di caratteri specificate dall'utente secondo alcune regole ed è molto potente e versatile. Nel Development package sono state inserite le funzioni di ricerca impiegate nella utility in modo che l'utente possa servirsene liberamente nei propri programmi.

LMK è una utility in tutto e per tutto simile alla utility "make" di UNIX; serve per stabilire le relazioni di dipendenza tra i file che costituiscono le varie parti di un progetto di grosse dimensioni, in modo tale che solo i moduli che sono stati cambiati e tutti quelli che dipendono da essi vengano ricompilati. Si rivela particolarmente comoda su sistemi dotati di hard disk, ma può

risultare molto utile anche in parecchi altre circostanze.

GO è l'ottimizzatore globale; è, in altri termini, un programma che tenta di migliorare le prestazioni di un programma e, come potete immaginare, viene prevalentemente usato su programmi scritti da altri, piuttosto che sui propri. L'ottimizzatore opera sui file quad e controlla l'assegnamento dei registri, la presenza di variabili che non vengono impiegate e così via. Analizza la disposizione del codice, individua sottoespressioni che ricorrono varie volte nel programma, rimuove i calcoli invarianti all'interno dei loop (ma che cosa ci stanno a fare lì dentro, dico io!) ed esaminano altre possibilità di evitare calcoli inutili. Alla fine ciò che ottenete è un programma che funziona meglio o al più nella stessa maniera di quello originale e, stando così le cose, non vedo che ragione ci possa essere per non usarlo.

LPROF e **LSTAT** costituiscono il cosiddetto profiler. LPROF individua le zone del programma a cui viene passato il controllo per la maggior parte del tempo e LSTAT elabora e rende leggibili i dati ottenuti da LPROF. Insieme, i due programmi possono fornire utili indicazioni su quali parti del programma possano trarre maggiore beneficio da speciali attenzioni. La Lattice deve aver usato parecchio queste due utility durante lo sviluppo della release 5.0.

SPLAT è un sostanzialmente un editor di linea; è simile per certi versi a Ed, benché un po' più sofisticato.

TOUCH modifica la data di creazione di un file; viene usato generalmente da programmi che modificano in tale maniera interi insiemi di file.

WC fornisce alcune statistiche sul contenuto di un file riguardanti il numero di caratteri, parole e linee presenti. Può anche calcolare un checksum sui soli caratteri stampabili. Di questa utility è fornito anche il sorgente in modo tale che possa essere portata su altri sistemi se necessario.

LSE, Lattice Screen Editor, è un altro programma molto comodo da usare. Esibisce caratteristiche specificamente rivolte ai programmatori C e assembler, come la presenza di un help online, dell'indentazione automatica e del ritorno a capo automatico. È possibile far eseguire automaticamente intere sequenze di operazioni associate a tasti e avere contemporaneamente aperti più file; inoltre sono disponibili molte funzioni di ricerca e sostituzione di stringhe. C'è anche la possibilità di definire delle macro per richiamare rapidamente le sequenze di comandi o le stringhe usate più di frequente.

Confrontando LSE con gli altri soliti editor, si evidenziano tre precisi vantaggi di questo editor rispetto alla concorrenza. In primo luogo, che vi piaccia o no, LSE è parte del package; in secondo luogo, è comodo perché sta insieme a tutte le altre utility e, infine, è possibile chiamare il compilatore da LSE ed ottenere gli eventuali messaggi d'errore senza lasciare l'editor.

Quest'ultima possibilità semplifica il ciclo di sviluppo di un programma e ne riduce sensibilmente i tempi morti. Bisogna poi notare, a completamento delle osservazioni precedenti, che LSE risulta ugualmente comodo per un programmatore assembler (vi ricordo che, nel kit di sviluppo, la Lattice ha messo anche un ottimo assemblatore e tutti i file .i necessari).

LCOMPACT è una utility per la compressione dei file header. Questo comando produce un file compattato in cui le parole riservate più comuni sono state codificate e gli spazi inutili e i commenti sono stati rimossi. Lcompact elabora anche la rappresentazione delle costanti numeriche con il risultato di ottenere file header di minori dimensioni e in una forma che il compilatore riesce a "digerire" più rapidamente.

Tutti i file Header forniti con il Lattice Development System 5.0 sono disponibili sia in forma compressa che non compressa (e leggibile dagli umani!).

C'è poi una gran quantità di programmi di supporto al programmatore; di questi, mi limiterò a menzionare la presenza di una utility di traceback, che serve per analizzare le cause di un crash (ovvero di una visita del Guru). Questa utility, in unione ad una speciale routine di startup, permette di recuperare dati messi sullo stack durante l'esecuzione, il contenuto dei vari registri e altre cose di questo genere, dopo che si sia verificata un'abend, cioè una terminazione anormale, del vostro programma.

Tra le altre utility, si distingue fd2pragma, che converte i file FD della Commodore in istruzioni pragma impiegabili nei propri programmi C.

Debugging a livello di sorgente

Per ultimo, certamente non in ordine di importanza, vorrei citare CodeProbe, il nuovo debugger a livello di sorgente della Lattice. Questo è senz'altro un programma molto interessante, perché permette di eseguire codice C compilato e di esaminarne il funzionamento a livello del sorgente.

Per poter utilizzare CodeProbe su un vostro programma, è necessario specificare una certa opzione al momento della compilazione, in modo che il linker includa nel modulo eseguibile tutte le informazioni necessarie al debugger per posizionare le linee di sorgente correttamente in relazione alle istruzioni del codice compilato e per accedere tramite il solo nome a tutti i simboli definiti nel programma.

Quando si fa eseguire il programma, la linea di sorgente interessata dall'esecuzione viene evidenziata. Le linee in cui c'è una chiamata a una funzione possono essere eseguite in un passo oppure si può entrare nella funzione e proseguire l'esecuzione passo-passo al suo interno. Si possono disporre breakpoint (anche condizionali), si possono esaminare i contenuti delle variabili, dei registri del 68000 e di zone della memoria.

Potete vedere contemporaneamente le linee di codice C e il loro equivalente in assembler e controllare i valori assunti dalle variabile accedendo ad esse per nome.

Continua a pagina 66

Introduzione a ROBBS

Rexx Object Building Blocks - un progetto ARexx

di Larry Philips

ARexx è uno strumento molto potente su Amiga, in quanto permette ai programmi di comunicare fra di loro abbastanza facilmente. In questo articolo, primo di una serie, Larry Philips descrive un'applicazione, ROBBS, che mette a disposizione una serie di blocchi, di mattoni da costruzione, per aiutare i programmatori a scrivere semplici programmi ARexx che controllino altri programmi esterni usati a loro volta come mattoni da costruzione. Questo articolo descrive il concetto generale e presenta SerMod (modulo seriale), il primo modulo della serie, progettato per permettere alla porta seriale di comunicare con programmi compatibili ARexx.

L'Amiga, per propria natura, incoraggia la scrittura di programmi che si allontanano dai modelli normali. Su alcune macchine troviamo word processor integrati con database manager, spreadsheet e programmi di mailing list. Gli autori di questi programmi fanno grandi sforzi per fornire una serie nutrita di scelte e di funzioni e per integrare in maniera coerente le varie parti del pacchetto.

Sui computer non multitasking troviamo, inoltre, tutta una serie di funzioni, contenute in programmi grandi e piccoli, che eseguono compiti molto semplici, quali la cancellazione di un file, la stampa del contenuto di una directory e così via. La ragione della presenza di queste funzioni è l'impossibilità di compiere tali operazioni a meno di non volere uscire dal programma in questione.

Se avete mai cercato un programma particolare, o una serie di programmi, per aggiornare in maniera remota un database, utilizzando un modem, vi sarete probabilmente rassegnati al fatto che gli unici programmi che avete trovato potevano svolgere questo lavoro solo in modo batch (esecuzione di una serie di comandi contenuti in un file), dopo la fine delle ore di attività lavorativa, mentre il vostro programma di database era inattivo.

Ebbene, noi possessori di Amiga sappiamo che c'è un modo migliore e abbiamo la macchina per risolvere questo tipo di problemi.

Per il fatto che Amiga è una macchina multitasking, non ci sono ragioni sufficienti perché un autore debba includere nel proprio programma quelle funzioni che sono svolte meglio da altri programmi. Il nostro programma di terminale, per esempio, non deve per forza incorporare una funzione per ottenere il contenuto di una directory, perché è così semplice passare nel CLI (o aprirne un altro) per eseguire l'operazione. In realtà, un nostro semplice programma di terminale è molto più potente, o perlomeno è potenzialmente molto più potente di qualsiasi altro analogo che possiate trovare su altre macchine. Possiamo lanciare virtualmente qualsiasi programma mentre siamo in linea, ottenendo un grado di libertà e di flessibilità che sulle macchine inferiori può essere solamente sognato.

Sebbene questa capacità sia molto importante, esiste un altro livello di versatilità che, fino a questo momento, è rimasto inesplorato. È bello poter eseguire più programmi nello stesso momento, ma non sarebbe ancora più bello se potessimo usare qualsiasi programma con qualsiasi altro programma, come se questi fossero stati scritti per lavorare assieme?

Oggi questo non è possibile, perlomeno non con *tutti* i programmi esistenti, ma potrebbe esserlo in futuro. Ho sentito un possessore di Amiga dire "database, word processor e animator tutti collegati insieme... potrebbero fare sparire Hypercard dalla vergogna." Personalmente non riesco a pensare a un utilizzo pratico di questi tre programmi integrati, ma il fatto è che qualcuno ha pensato che ciò sarebbe meraviglioso.

Dubito che l'autore di un qualsiasi database possa anche solo considerare l'eventualità di scrivere il codice per permettere al suo programma di scambiare dati con gli altri due e la stessa cosa vale per gli altri due autori. Anche se questi pensassero che è una buona idea, quale programma dovrebbero scegliere per mettere in pratica il collegamento? Se il prescelto fosse un programma a noi particolarmente gradito ne saremmo felici, ma in caso contrario continueremmo ad augurarci che avessero scelto il nostro favorito.

Questi pensieri mi hanno portato all'idea che quello di cui abbiamo realmente bisogno sono una serie di mattoni da costruzione progettati per essere collegati insieme e contenenti ognuno una serie, limitata ma necessaria, di funzioni.

Non posso, comunque, prendermi il merito per l'idea di base.

Questa è una filosofia che ha resistito alla prova del tempo in un buon numero di sistemi operativi. UNIX è uno di questi, e, anche se non lo amate in modo particolare, dovete comunque ammettere che la possibilità di costruire programmi complessi partendo da programmi piccoli e semplici è un'idea molto valida.

Ad ogni modo, personalmente voglio andare un gradino più in là e scrivere una serie di moduli che possano essere controllati più in dettaglio. Mentre Unix permette la manipolazione dei programmi attraverso *stdin*, *stdout*, pipe, redirezione e file intermedi, io intendo entrare nell'essenza dei piccoli programmi e permetterne una manipolazione più diretta.

I messaggi fra programmi

Amiga dispone di una funzione interna che risulta ideale per i nostri scopi. Può inviare messaggi fra programmi attraverso un sistema di porte apposite (message port).

Il funzionamento di una message port è molto semplice. Un programma può creare una message port, assegnarle un nome e renderla pubblica inserendola in una lista di sistema. Una volta fatto questo, qualsiasi altro programma può cercare una porta con un certo nome e, in caso questa sia trovata, spedirle un messaggio.

Il messaggio viene ricevuto dalla porta e il programma che ha creato la porta può farne ciò che desidera. Il programma ricevente deve essere, naturalmente, in grado di comprendere il messaggio. A questo punto il ricevente risponde al messaggio per fare sapere al programma mittente che il suo messaggio è stato ricevuto e per permettere che il mittente riutilizzi come meglio crede la zona di memoria occupata dal messaggio stesso.

I messaggi e ARexx

Prima dell'avvento di ARexx, la capacità di Amiga di scambiare messaggi era limitata a programmi compilati in una serie di linguaggi differenti e quindi fuori portata per molti utenti. ARexx, invece, sfrutta appieno questa potenzialità e può mandare messaggi ad altri programmi ARexx (script) oppure a programmi (eseguibili) progettati per comprendere i messaggi di ARexx.

Questo apre nuovi orizzonti a coloro che amano i linguaggi interpretati, permettendo la scrittura di semplici programmi (non così semplici) ARexx che chiamano altri programmi come mattoni da costruzione, in una sorta di *construction set* per produrre applicativi.

Dopo mesi di discussioni con altri programmatori su questa idea e non essendo capace di interessarli a sufficienza tanto da convincerli a iniziare a scrivere i moduli, ho cominciato a scriverne qualcuno personalmente.

Ho chiamato questa serie di moduli ROBBS, abbreviazione di Rexx Object Building Block System (sistema Rexx di costruzione a mattoni).

Il modulo seriale SerMod

Il primo modulo della serie si chiama *SerMod* (da Serial Module). È stato progettato per essere piccolo (circa 12K, ma può essere reso ancora più piccolo) e per contenere un numero limitato di funzioni che sono indispensabili esclusivamente per l'uso della porta seriale.

Quando è nato era una combinazione di funzioni per la porta seriale e per il display, ma appena la parte seriale ha incominciato a lavorare in modo soddisfacente, ho separato la parte che riguardava il display e con quella ho creato un altro modulo chiamato (avete indovinato) DispMod. DispMod è un semplice programma di display che dispone di un solo tipo di finestra e di una funzionalità limitata all'elaborazione degli input e alla gestione degli output.

SerMod in sé stesso è abbastanza limitato. Potete lanciarlo da uno script e mandargli dei comandi da quello stesso script per compiere una serie di azioni disparate, quali settare i parametri della porta seriale, spedire del testo, ricevere del testo e confrontare i dati entranti con le stringhe fornite dall'utente.

Con uno script appropriato potrebbe funzionare come utility per il trasferimento di file, come una BBS o come un programma per permettere operazioni tipo CLI a distanza (in maniera limitata).

La combinazione di uno script, di SerMod e di DispMod diventa, invece, un programma di terminale abbastanza potente. Quando dico abbastanza potente, mi riferisco alle possibilità che si hanno con uno script di ARexx, dal momento che il modulo dei protocolli, il modulo dei menu, il modulo dei gadget e così via devono ancora essere implementati!

Ho scelto di scrivere prima SerMod perché molti programmi possono trarre beneficio dalla possibilità di effettuare comunicazioni attraverso la porta seriale e perché non è difficile immaginare i modi in cui si può combinare un modulo seriale con altri programmi.

SuperBase viene adesso fornito con un'interfaccia ARexx, così come MicroFiche Filer. Anche CygnusEd Professional, TxEd, Uedit, DME, PCLO e PCLO Plus sono dotati di interfaccia ARexx.

Con SerMod e uno script ARexx, uno qualsiasi di questi programmi acquista improvvisamente la capacità di comunicare attraverso la porta seriale, via cavo in ambito locale e via modem in ambito remoto. Se si crea abbastanza interesse in questo progetto, verranno indubbiamente scritti molti altri moduli, piccoli programmi che fanno solo poche operazioni e che ci permetteranno di costruire altri applicativi. Anche coloro che non desiderano programmare, nemmeno in ARexx, possono trarre beneficio dai programmi ARexx scritti per ROBBS.

Cos'è esattamente ROBBS?

Forse è il caso di dire due parole sulla filosofia che sta dietro a ROBBS.

Una tipica applicazione di ROBBS consisterà in uno o più moduli ROBBS, controllati da uno o più script ARexx. In aggiunta, potrebbero essere chiamati anche dei programmi esterni, sia che abbiano un'interfaccia ARexx o meno. Qualsiasi programma che può fornirci i dati che ci interessano, magari in un file, può essere usato in questo contesto.

Ogni modulo ROBBS usato in un applicativo dovrà compiere solo poche operazioni, per mantenere limitate le dimensioni, per semplificare la programmazione e per assicurare che ci sia il minor spreco di sforzo che usualmente si verifica quando vengono implementate le stesse funzioni in moduli diversi. Solo quei comandi che hanno senso in quel determinato contesto devono essere creati.

Per il motivo precedente, non c'è bisogno di definire una serie di comandi standard che accomunino tutti i moduli, anche perché è abbastanza facile modificare i programmi di controllo ARexx per adeguarsi alle eventuali differenze. Ci sono alcuni comandi che hanno senso in tutti i moduli e questi dovrebbero essere chiamati tutti con lo stesso nome. Alcuni esempi sono STATUS, CONNECT, CTRL e DIE (ne parleremo più avanti).

A questo punto vorrei darvi l'idea di come dovrebbe apparire il tipico modulo dal punto di vista di un programmatore ARexx. Utilizzerò SerMod come esempio. Sebbene sia già stato scritto, non è 'scolpito nella pietra'. Se avete qualsiasi suggerimento per l'aggiunta di altre funzioni, non esitate a contattarmi.

I comandi SerMod

RXD abilita o disabilita la ricezione di dati seriali.

Permette agli altri programmi che aprono la porta seriale in modo SHARED di avere accesso ad essa senza interferenze.

Accetta ON o OFF come argomenti.

TXD abilita o disabilita la trasmissione di dati seriali.

Permette a un altro programma di spedire dati senza alcuna interferenza da parte di SerMod. L'altro programma deve aprire la porta seriale in modo SHARED.

Accetta ON o OFF come argomenti.

SEND manda una stringa sulla porta seriale.

Una variante di questa funzione, chiamata LSEND, manda una stringa facendola seguire da un carattere di Carriage Return.

Accetta una stringa come argomento.

MATCH prepara e passa a SerMod una stringa di paragone che deve essere ricercata nei dati che arrivano dalla porta seriale.

Quando SerMod trova questa stringa nel flusso di dati entrante, manda un messaggio alla porta CTRL, specificando il numero della stringa che è stata trovata. SerMod permette infatti la definizione di 10 stringhe di 100 caratteri l'una. Il comando MATCH può essere eseguito quante volte si vuole, per permettere la sostituzione delle stringhe di paragone specificate in precedenza. MATCH permette anche la cancellazione selettiva o totale delle stringhe di paragone. Notate che la funzione SCAN deve essere attiva affinché il comando MATCH abbia effetto.

Accetta un numero compreso fra 0 e 9, seguito da una stringa. Per esempio:

```
MATCH 3 'Questa è una stringa'
```

SCAN abilita la funzione di ricerca delle stringhe di paragone preparate con MATCH.

Accetta ON o OFF come parametri.

CONNECT collega il flusso di dati ricevuti a una specifica message port.

Permette anche di stabilire il comando che verrà usato per spedire i dati, dal momento che non tutti i moduli utilizzano lo stesso. Si possono mandare i dati entranti allo script di controllo, oppure direttamente a un altro modulo ROBBS.

Accetta due argomenti, il nome di una porta e un comando.

CTRL collega SerMod a un'altra porta per motivi di controllo.

Qualsiasi programma può mandare messaggi a SerMod, mentre quest'ultimo manda il flusso di dati entranti alla porta CONNECT e le eventuali stringhe di paragone trovate nel flusso alla porta di controllo CTRL. Questo significa che possiamo passare il controllo da uno script a un altro script, come meglio crediamo.

Accetta il nome di una porta come argomento.

COMM modifica i parametri della porta seriale.

Permette di stabilire il baud rate, il modo (echo, noecho), lo handshaking e il formato dei dati (7 o 8 bit).

Gli argomenti di questa funzione verranno spiegati meglio in una successiva trattazione dettagliata dei comandi.

DIE libera le risorse allocate e rimuove SerMod dal sistema.

Se tutti i messaggi spediti da SerMod hanno ricevuto una risposta, questa operazione viene effettuata immediatamente. Se c'è ancora qualche messaggio che non ha ricevuto risposta, viene stampato un avviso nel CLI dal quale SerMod è stato lanciato e l'utente deve dare il comando REALLY_DIE per rimuovere SerMod. Questo eventualità non dovrebbe mai accadere in un applicativo già testato e corretto e viene fornita solo come uscita di emergenza per quando uno script si interrompe a causa di un errore.

Non richiede argomenti.

CGSS

SE CERCHI IL TUO MIGLIORE AMICO, CERCALO IN UN CANILE.

E di amici a quattro zampe ne troverai non uno, ma migliaia. Sono i cani abbandonati ospitati presso i Canili della Lega. Cani che un tempo avevano un nome e un padrone, cani che adesso hanno solo paura. Paura di finire i loro giorni dietro le sbarre, senza mai più sentire la carezza di un uomo. Perciò, se cerchi un amico, cercalo in un canile: ti sta aspettando. Per maggiori informazioni telefona allo 010/561557. Se invece non puoi adottarne uno, puoi fare comunque molto per loro, inviando un'offerta in denaro sul CCP 17182122. Il tuo aiuto servirà a tenere in vita la speranza che un giorno possa ricominciare una storia d'amore senza fine: quella tra l'uomo e il suo cane.

CCP 17182122 · UFFICIO PROPAGANDA E SVILUPPO · VIA GIANOLIO 31/4 12042 BRA

TEL. 010/561557

STATUS restituisce una stringa contenente lo stato attuale di SerMod.

La stringa contiene informazioni sulle regolazioni della porta seriale, sullo stato di SCAN, TXD e RXD, sulle porte CONNECT e CTRL e, per usi futuri, contiene l'indirizzo della struttura IOSerRequest che SerMod sta utilizzando.

Non richiede argomenti. Il formato della stringa restituita verrà descritto in una successiva trattazione dettagliata dei comandi.

Articoli futuri

Prevedo di pubblicare il codice sorgente di SerMod in uno dei prossimi numeri, insieme ad alcuni esempi di programmi ARexx e alle spiegazioni di come usare SerMod.

Nel frattempo ponderate quanto detto in questa puntata, in modo particolare riguardo la versatilità del collegare più programmi attraverso ARexx e lasciate correre la vostra immaginazione.

Nei mesi a venire scriverò almeno uno o due altri moduli, magari anche di più, e presto saremo ben avviati sulla strada che ci porterà a disporre di alcuni dei più veloci ed efficienti applicativi che possiate immaginare.

(segue da pagina 61)

Lattice C 5.0

Le opzioni sono così tante che potremmo neanche sperare di menzionarle tutte; per rendersene conto, basta osservare che la documentazione relativa a CodeProbe costituisce circa un terzo del secondo manuale.

Bisogna ammettere che CodeProbe è un programma che si usa in maniera molto piacevole e che, al tempo stesso, può soddisfare anche gli utenti professionali. Non si può certo affermare con assoluta certezza che sia privo da errori e da problemi di varia natura, anzi è probabile che con la diffusione del suo uso se ne evidenzino più d'uno, ma tutto concorre a dare l'impressione che il programma rimarrà comunque molto utile.

Conclusioni

In definitiva, la versione 5.0 del Lattice Development System è un pacchetto software dalle caratteristiche molto professionali. Certamente procurerà molti nuovi clienti alla Lattice, soprattutto per il buon rapporto prestazioni-prezzo. Non che sia particolarmente economico, intendiamoci; piuttosto, bisogna considerare il fatto che viene fornita una gran quantità di software e una eccellente documentazione e non credo che si possa dire che il prezzo è elevato per tutto quello che viene offerto. Gli utenti registrati Lattice potranno poi ottenere la nuova release per un prezzo molto inferiore al valore del solo CodeProbe.

(segue da pagina 28)

Breakpoint

MS permette di mischiare la visualizzazione di linee del codice sorgente con il loro disassemblato in memoria, in modo da permetterci di vedere cosa fa il nostro compilatore. Selezionare una linea del codice sorgente è come selezionare l'indirizzo della prima istruzione del codice generato per quella linea.

Ebbene, abbiamo più o meno visto quali sono le differenze fra i debugger. Che vi piacciano le finestre e il poter puntare e selezionare con il mouse o che preferiate lavorare in un ambiente testuale, questo dipende dal vostro gusto personale. Io penso che l'avere una zona di memoria costantemente visualizzata sia una caratteristica estremamente utile, ma sono sicuro che ci siano altri che preferiscono l'esecuzione di 'liste di comandi' a ogni breakpoint.

In conclusione

Spero che vi sia piaciuto leggere questa serie di articoli. Io mi sono divertito a scriverli. Forse dopo che MetaScope verrà reso *completamente* compatibile con il codice sorgente e l'Avant-Garde avrà finito il suo debugger M2, daremo ancora un'occhiata alle innovazioni presenti nel campo del debugging e alle sue ultime novità. Presumo che Lattice e Manx avranno, per allora, prodotto delle versioni più avanzate dei loro CPR e SDB.

Ho tentato di non essere troppo dogmatico in questa serie di articoli. La mia esperienza con molti sistemi, debugger e programmatori mi dice che ogni persona deve trovare il suo stile, quello con il quale si trova meglio. Il debugging è ancora un'arte in molti sensi della parola e se *voi* vi trovate bene con una determinata tecnica, allora per voi funzionerà benissimo. Se *non* siete contenti di quella tecnica, potrà solo ostacolarvi.

Vorrei lasciarvi con qualcosa che mi ha aiutato più di ogni altra cosa nel mio debugging:

Abbiate sempre *qualche* idea di che cosa vi state aspettando *prima* di fare una prova.

```
Egli saltò alla sua tastiera
    Un ultimo break da inserire
Ma cominciò a svanire
    Sembrava molto sottile

E lo sentii esclamare
    prima che sparisse di vista
'Buon debugging a voi tutti...
    Fategliela vedere a quel GURU!'
```

I device handler

Come creare un device handler per AmigaDOS

di Steve Simpson

Steve Simpson è un esperto di software in ambito Amiga e PC. Un ex-astronomo, Steve ora traduce documentazione tecnica dallo svedese; i suoi attuali interessi riguardano lo sviluppo di sistemi e di firmware. Al momento si sta occupando del miglioramento delle routine di interfacciamento del SideCar, relativamente al pacchetto software che dovrebbe essere disponibile alla fine del 1989. Steve è anche l'autore di DiskRepair, una utility per l'editing dei dischi e il recupero di file danneggiati.

Tutti coloro i quali impiegano il CLI o il WorkBench saranno certamente familiari con l'AmigaDOS e il filing system. Sia il CLI sia il WorkBench fanno per forza di cose un uso intensivo dei comandi messi a disposizione dall'AmigaDOS. Il nome di "filing system" si applica solamente ai dispositivi che appaiono con un'icona nella finestra del WorkBench e che generalmente corrispondono a disk drive; ci sono, però, anche altri tipi di dispositivi, detti "non-filing", disponibili sotto AmigaDOS, come PRT:, SER:, RAW: e CON:. L'insieme dei "device" disponibili non è fisso e può essere ampliato secondo le necessità.

Questo articolo è il terzo di una serie che ha avuto origine da un progetto di grandi dimensioni nel quale sono attualmente impegnato; intende trattare il progetto e l'implementazione di un device handler in AmigaDOS e ha lo scopo di chiarire un argomento che, nella documentazione ufficiale, viene trattato in maniera molto approssimativa.

Che cos'è un device di Amiga?

Il termine "device" nell'ambiente Amiga si riferisce a due oggetti distinti. Ci sono device Exec, quelli cioè che troviamo nella directory DEVS: con il suffisso .device, a cui è affidato il compito di occuparsi delle richieste di I/O verso i dispositivi hardware veri e propri (device Exec tipici sono il keyboard.device e il trackdisk.device) e ci sono i device DOS, altrimenti detti handler.

Per evitare malintesi, sappiate che, in questo articolo, la dizione system device si riferisce a un device Exec, mentre i termini DOS device, DOS device handler e DOS handler sono tutti sinonimi.

I device DOS forniscono al sistema la possibilità di interagire in maniera omogenea con i differenti dispositivi del mondo esterno. I device DOS vengono trattati come se fossero file, impiegando chiamate a funzioni DOS come Open(), Close(), Read() e Write(); è poi compito dello specifico handler tradurre queste chiamate nelle opportune sequenze di operazioni, specifiche della particolare implementazione. Più precisamente, un handler rappresenta in generale, l'interfaccia tra il DOS e un device Exec; un device DOS può, infatti, servirsi dei device Exec per effettuare le operazioni di I/O richieste e, in effetti, questo è ciò che succede nella maggior parte dei casi (ad es. SER: apre il serial.device e DF0: usa il trackdisk.device):

Un device DOS viene aperto con la chiamata:

```
file_handle = Open(nome_di_device,modo);
```

dove:

file_handle è un puntatore a una struttura FileHandle, definita in libraries/dosextens.h

nome_di_device è una stringa di caratteri che specifica il nome del device (CON:, RAW:, etc.)

modo è un intero long che specifica il modo; MODE_NEWFILE apre un device sia per la lettura sia per la scrittura.

La chiusura di un device precedentemente aperto viene effettuata con la funzione

```
Close(file_handle);
```

I pacchetti e l'AmigaDOS

Le comunicazioni tra i programmi applicativi e il DOS avvengono tramite l'invio di pacchetti, grazie alle possibilità di "message passing" messe a disposizione dall'Exec. Un pacchetto è una struttura costruita estendendo la struttura Message dell'Exec ed è definita in dosextens.h nella seguente maniera:

```
struct  DosPacket
  {
  struct    Message      *dp_Link;
```

```
struct    MsgPort      *dp_Port;
LONG                   dp_Type;
LONG                   dp_Res1;
LONG                   dp_Res2;
LONG                   dp_Arg1;
LONG                   dp_Arg2;
LONG                   dp_Arg3;
LONG                   dp_Arg4;
LONG                   dp_Arg5;
LONG                   dp_Arg6;
LONG                   dp_Arg7;
};
```

Il formato di un pacchetto dipende dal tipo e da quest'ultimo dipendono anche gli argomenti (Arg1 - Arg7) effettivamente usati. Un'applicazione scritta dall'utente può anch'essa inviare pacchetti a un handler, ma non ci occuperemo di questo aspetto in questo articolo.

I pacchetti DOS vengono inviati al message port dello handler, port che è automaticamente creato dal sistema come parte della struttura che descrive il processo. Il message port è inizializzato in modo che l'arrivo di un messaggio causi l'attivazione del segnale 8. Questo fa sì che si possa richiamare la funzione di Exec Wait() (relativamente al segnale 8) per attendere l'arrivo di un pacchetto dal DOS.

Con la funzione GetMsg() si rimuove il messaggio dal port; l'indirizzo del pacchetto viene, quindi, preso dal campo ln_Name della struttura Node contenuta nel messaggio; il tipo di azione da effettuare viene invece specificato nel campo dp_Type del pacchetto.

Quando lo handler ha terminato di compiere le operazioni richieste dal pacchetto, lo restituisce attraverso una chiamata a PutMsg(), passando la medesima struttura.

Al momento attuale ci sono almeno due dozzine di tipi di pacchetto ufficialmente definiti, ma non tutti possono essere mandati a qualunque handler. Inoltre, è anche opportuno osservare che gli argomenti assumono un significato diverso a seconda del tipo di pacchetto.

Res1 viene posto a zero di norma per segnalare un errore e in tal caso Res2 fornisce maggiori dettagli in merito al tipo dell'errore (il valore di Res2 è quello che si ottiene chiamando la funzione IoErr()). Un -1 in Res1, invece, indica il completamento con successo delle operazioni richieste qualora il pacchetto inviato usi questo campo per comunicare la presenza di errori.

Gli argomenti possono essere sia interi long sia puntatori BCPL e in quest'ultimo caso possono presentarsi sia puntatori a stringa (BSTR) che puntatori a struttura (BPTR). Le differenze tra un puntatore C e uno BCPL saranno esaminate in dettaglio più avanti.

La **tavola 1** descrive brevemente l'impiego e il significato dei vari campi della struttura DosPacket al variare del tipo (specificato in dp_Type). La maggior parte dei tipi di pacchetto è definita nel file libraries/ dosextens.[hi]; gli altri tipi di pacchetto ufficialmente riconosciuti sono comunque riportati nel file new-handler.h, che potete trovare al termine di questo articolo insieme con gli altri listati.

Le strutture dati del DOS

Il DOS impiega un certo numero di strutture dati per i propri scopi; la definizione di tali strutture è riportata nel file libraries/dosextens.[hi]. Esaminiamole in dettaglio.

```
struct   DosLibrary
  {
  struct   Library      dl_lib;
  APTR                  dl_Root;
  APTR                  dl_GV;
  LONG                  dl_A2;
  LONG                  dl_A5;
  LONG                  dl_A6;
  };
```

dl_lib: una struttura Library, che costituisce un nodo della lista delle library presenti nel sistema.

dl_Root: un puntatore alla struttura RootNode.

dl_GV: un puntatore al "global vector" (usato col BCPL).

dl_A2, dl_A5 e dl_A6: campi usati per tenere copie temporanee dei registri.

La struttura RootNode è definita nella maniera seguente:

```
struct   RootNode
  {
  BPTR                  rn_TaskArray;
  BPTR                  rn_ConsoleSegment;
  struct   DateStamp    rn_Time;
  LONG                  rn_RestartSeg;
  BPTR                  rn_Info;
  BPTR                  rn_FileHandlerSegment;
  };
```

rn_TaskArray: un array che specifica i processi CLI già attivi.

rn_ConsoleSegment: la SegList relativa al CLI.

rn_Time: l'ora attuale.

rn_RestartSeg: la SegList per il "disk validator".

rn_Info: un puntatore BCPL a una struttura DosInfo.

La struttura DosInfo è così definita:

```
struct   DosInfo
  {
  BPTR         di_McName;
  BPTR         di_DevInfo;
```

```
    BPTR        di_Devices;
    BPTR        di_Handlers;
    APTR        di_NetHand;
    };
```

di_McName: il nome di questa macchina nella rete (per il momento questo campo è posto a zero).

di_DevInfo: un puntatore BCPL alla lista dei device e dei volumi disponibili.

di_Devices: per il momento, zero.

di_Handlers: per il momento, zero.

di_NetHand: il process ID dello handler della rete.

L'AmigaDOS si preoccupa di gestire e mantenere aggiornata una lista di strutture che descrivono i "device" fisici, quelli virtuali (quelli, cioè, creati con ASSIGN) e i vari volumi disponibili; per specificare a quale di questi tre tipi di oggetti si riferisce una certa struttura della lista viene impiegato un campo che può assumere uno tra i valori DLT_DEVICE, DLT_DIRECTORY e DLT_VOLUME rispettivamente.

Il formato di un elemento della lista dipende dal tipo di oggetto a cui tale elemento si riferisce. Un device o una directory viene descritta con la struttura DevInfo, che è così costituita:

```
struct   DevInfo
  {
    BPTR        dvi_Next;
    LONG        dvi_Type;
    APTR        dvi_Task;
    BPTR        dvi_Lock;
    BSTR        dvi_Handler;
    LONG        dvi_StackSize;
    LONG        dvi_Priority;
    LONG        dvi_Startup;
    BPTR        dvi_SegList;
    BPTR        dvi_GlobVec;
    BSTR        dvi_Name;
    };
```

Il significato dei campi è il seguente:

dvi_Next: puntatore al prossimo elemento della lista.

dvi_Type: tipo dell'oggetto a cui questa struttura si riferisce; in questo caso può essere solo DLT_DIRECTORY (1L) o DLT_DEVICE (0L).

dvi_Task: process ID del processo dello handler, se abbiamo a che fare con un device, altrimenti zero.

dvi_Lock: puntatore BCPL a un lock del file system oppure zero.

dvi_Handler: nome del file dello handler oppure zero.

dvi_StackSize: dimensioni dello stack associato al processo dello handler.

dvi_Priority: priorità del processo dello handler.

dvi_Startup: valore di inizializzazione da passare al processo dello handler.

dvi_SegList: puntatore BCPL alla "segment list" del processo dello handler oppure zero.

dvi_GlobVec: puntatore BCPL al "global vector" del processo dello handler oppure zero.

dvi_Name: nome del device fisico o virtuale.

Se il campo dvi_Type contiene il valore DLT_VOLUME (2L), l'elemento della lista descrive un volume e viene impiegata la seguente struttura:

```
struct   DeviceList
  {
    BPTR                      dl_Next;
    LONG                      dl_Type;
    struct      MsgPort       *dl_Task;
    BPTR                      dl_Lock;
    struct      DateStamp     dl_VolumeDate;
    BPTR                      dl_LockList;
    LONG                      dl_DiskType;
    LONG                      dl_unused;
    BSTR                      dl_Name;
    };
```

in cui i campi hanno il seguente significato:

dl_Next: puntatore al successivo elemento della lista.

dl_Type: tipo dell'oggetto a cui si riferisce questa struttura; in questo caso non può essere che DLT_VOLUME (2L).

dl_Task: process ID dello handler del dispositivo su cui è montato il volume a cui si riferisce la struttura, altrimenti zero.

dl_Lock: puntatore BCPL ad un lock del file system.

dl_VolumeDate: data di creazione del volume.

dl_LockList: puntatore BCPL alla lista di lock attivi relativi a questo volume.

dl_DiskType: tipo del disco su cui risiede il volume.

dl_Name: puntatore a stringa BCPL che specifica il nome del volume.

La struttura DevInfo è equivalente alla struttura DeviceNode definita in libraries/filehandler.[hi]:

```
struct   DeviceNode
  {
```

```
BPTR                  dn_Next;
ULONG                 dn_Type;
struct    MsgPort     *dn_Task;
BPTR                  dn_Lock;
BSTR                  dn_Handler;
ULONG                 dn_StackSize;
LONG                  dn_Priority;
BPTR                  dn_Startup;
BPTR                  dn_SegList;
BPTR                  dn_GlobalVec;
BSTR                  dn_Name;
};
```

I puntatori BCPL

Il BCPL è stato un precursore del C che consentiva l'uso di un solo tipo di variabile: la long word. Per questa ragione tutto in BCPL veniva allocato a partire da indirizzi multipli di quattro (il numero di byte di una long word).

È importante ricordare, quando dovete allocare una stringa BCPL o un qualsiasi altro oggetto a cui si debba accedere con un puntatore, che usando la funzione della libreria Exec AllocMem() si può star tranquilli che il blocco ottenuto soddisfi la condizione di allineamento a un multiplo di quattro dalla quale in BCPL non si sfugge.

Dal momento che il BCPL usa solo long word, anche i puntatori sono long word e vengono detti BPTR. Ciò che complica la situazione è che non sono semplici puntatori perché non contengono l'indirizzo effettivo in memoria, ma l'indirizzo diviso per 4. Potete ora comprendere che è assolutamente obbligatorio allocare i blocchi di memoria in modo che il loro indirizzo sia un multiplo intero di 4.

Le stringhe BCPL sono diverse dalle stringhe C. La stringa BCPL, detta più brevemente BSTR, è una struttura in cui il primo byte contiene la lunghezza della stringa e i successivi i caratteri della stringa stessa.

In generale si accede a una stringa BCPL tramite un BPTR. Per ottenere da un puntatore C l'equivalente puntatore BCPL bisogna "shiftare" il valore del puntatore di due bit verso destra; analogamente, per convertire da BCPL si "shifta" a sinistra di due bit. Le macro CPTRtoBPTR() e BPTRtoCPTR(), impiegate nell'esempio che accompagna questo articolo, effettuano le conversioni richieste.

I lock

A un processo che presenta una richiesta di apertura di un file viene generalmente concesso un certo tipo di accesso esclusivo al file in questione. L'accesso può essere allora condiviso o esclusivo del processo che ha fatto la richiesta. In tal caso il sistema passa al processo un cosiddetto "lock" sul file. Un lock rappresenta un identificatore univoco di un file ed è una forma più semplice di descrittore di file; può essere ottenuto nella maniera seguente:

```
file_lock = Lock(file_name,mode);
```

dove:

file_lock è un puntatore a una struttura FileLock, definita nel file libraries/dosextens.h. Viene restituito uno 0L se si è verificato un errore.

file_name è una stringa C contenente l'intero path e il nome del file.

mode specifica il tipo di accesso richiesto:

```
ACCESS_READ fornisce accesso condiviso,
ACCESS_WRITE fornisce accesso esclusivo.
```

La funzione

```
UnLock(file_lock)
```

rilascia un lock precedentemente ottenuto.

La struttura FileLock è così definita:

```
struct   FileLock
  {
  BPTR                 fl_Link;
  LONG                 fl_Key;
  LONG                 fl_Access;
  struct    MsgPort    *fl_Task;
  BPTR                 fl_Volume;
  };
```

Il significato dei membri che possono interessare all'utente è il seguente:

fl_Key: il numero del blocco sul disco a cui ha inizio il file.

fl_Access: tipo di accesso al file (ACCESS_READ o ACCESS_WRITE).

fl_Volume: puntatore al descrittore del volume associato al file o alla directory per la quale abbiamo richiesto il lock.

L'interfacciamento con i device dell'Exec

Come una qualsiasi altra applicazione, un handler può aver libero accesso a tutte le risorse del sistema. È molto comune che un handler si avvalga dei device dell'Exec, che, come abbiamo già precisato, si pongono tra il dispositivo fisico vero e proprio e le applicazioni. Per usare un device Exec, bisogna in primo luogo allocare e inizializzare una struttura per effettuare le richieste di I/O e creare un "reply port" (cioè un message port tramite il quale ottenere le risposte dal device); fatto questo, è possibile chiamare la funzione OpenDevice() per ottenere l'accesso al device desiderato. Quando si termina di usare il device, è opportuno rilasciare tale risorsa con la funzione CloseDevice().

I comandi vengono impartiti al device specificandone il tipo desiderato nella struttura di richiesta delle operazioni (e ovviamente inizializzando opportunamente gli altri campi) e chia-

Fondamentali per lo studio, il lavoro e l'aggiornamento

i dizionari enciclopedici di:

mando la funzione DoIO() o la funzione SendIO(), a seconda che si desideri un controllo sincrono o asincrono dell'esecuzione del comando. È comunque opportuno che vi riferiate all'articolo relativo ai device apparso sul numero 4 di Transactor e alla documentazione ufficiale per tutti i dettagli implementativi.

Un handler deve preoccuparsi di effettuare le operazioni di inizializzazione di ogni dispositivo che adopera e, in generale, deve riconoscere e rispondere almeno a un insieme ristretto di tipi di richieste, quali l'apertura, la lettura, la scrittura e la chiusura di un file. Inoltre, in certi casi è necessario che lo handler riconosca ed esegua anche altri tipi di richieste, come, ad esempio, SEEK, EXAMINE_OBJECT, CREATE_DIR ed altre nel caso di un handler per un filing system completo.

La segnalazione degli errori

Abbiamo visto che la struttura DosPacket contiene due campi, Res1 e Res2, che vengono usati per segnalare gli eventuali errori che si siano verificati nel tentativo di soddisfare la richiesta specificata nel pacchetto. È necessario che trattiate questi campi in maniera conforme alle convenzioni del DOS.

Quando il campo Res1 è posto a 0, viene segnalato un errore e Res2 contiene un valore che fornisce ulteriori informazioni sul tipo dell'errore che si è verificato. Il valore di Res2 può essere ottenuto con la funzione IOErr() della dos.library. Il file libraries/dosextens.[hi] contiene i codici di errore definiti a livello ufficiale, ma potete comunque definire dei vostri codici particolari per gli specifici errori che possono aver luogo nel vostro handler.

L'implementazione di un handler

Usualmente, un handler viene impiegato per rendere un device Exec disponibile al DOS, ma non deve essere necessaria la preesistenza di un device Exec per realizzare un handler; è, infatti, possibile immaginare l'uso di un handler in casi che non richiedano affatto l'interazione con un device Exec (si veda, ad es., il programma "ID-handler" del Fish disk #87).

Quando l'AmigaDOS installa un handler nel sistema, in risposta al comando Mount, esso crea una struttura DeviceInfo o DeviceNode, a seconda che si tratti di un handler per un device "filing" o "non-filing", e inserisce la struttura, dopo averne opportunamente riempito i campi, nella lista di sistema; dopodiché carica il codice dello handler in memoria e gli crea un processo apposito.

Nel caso in cui si stia installando un device "non-filing", è necessario che lo handler stesso ponga nel campo dn_Task un puntatore al proprio task (più precisamente, alla struttura MsgPort associata al task). Così facendo, infatti, l'AmigaDOS eviterà di creare un nuovo processo handler ogni volta che trova un riferimento al device a cui lo handler è associato, operazione che compierebbe, invece, se il campo dn_Task fosse posto a zero.

Quando l'AmigaDOS fa partire il processo dello handler, invia allo handler stesso un pacchetto di startup, in cui il campo Arg3 contiene un puntatore BCPL alla struttura DeviceNode associata allo handler.

Le prime versioni del sistema operativo imponevano l'uso del linguaggio BCPL per la realizzazione degli handler; fortunatamente, ora è possibile scrivere gli handler in C o in assembler, avendo la precauzione di specificare nella Mountlist -1 come GlobVec (il che segnala al DOS che lo handler non è in BCPL).

L'installazione dello handler NEW:

I moduli BCPL vengono "linkati" run-time attraverso il "global vector" che è una tavola comune di indirizzi, mediante la quale è possibile accedere alle variabili globali e effettuare una comunicazione tra moduli. In AmigaDOS i vari componenti vengono tenuti insieme da un global vector e ogni processo dispone di un global vector privato per i vari usi interni.

Affinché un nuovo device DOS possa essere riconosciuto dal sistema e possa essere fatto partire lo handler relativo (che sta nella directory L:), è necessario che sia presente nella Mountlist una sommaria descrizione del nuovo device: in tale descrizione, possiamo specificare il nome del file che ospita il codice dello handler, le dimensioni dello stack del processo dello handler e la priorità.

Nel nostro esempio, il file batch do_mount si occupa di far tutte le operazioni necessarie per rendere immediatamente disponibile il nuovo handler NEW:. Si noti che è necessario specificare specificare in quale directory sono presenti sia il codice dello handler sia la mountlist che contiene la descrizione di NEW:.

Dopo aver eseguito il comando

```
execute do_mount <directory>
```

potrete verificare con il comando assign che NEW: compare tra i device disponibili. Un handler per un device "filing" dovrebbe anche creare è inserire nella lista di sistema una struttura che descriva un volume associato al device; solo in tal caso, infatti, apparirà l'icona relativa nella finestra del Workbench.

La prima volta che si dà un comando che accede a NEW:, il suo handler viene caricato e fatto partire e dovrebbe apparire una finestra in cui vengono stampati dei messaggi relativi all'attività dello handler.

Che cosa fa NEW:

L'esempio di handler proposto ha unicamente scopi didattici; infatti, riconosce e risponde solo a un limitato numero di pacchetti e precisiamente a: OPENRW, OPENOLD, OPENNEW, READ, WRITE, CLOSE e DIE. Di conseguenza, le sole funzioni il cui impiego con NEW ha un senso sono: Open(), Close(), Read() e Write(). Il modulo test_new.c illustra come aprire, usare e chiudere il device NEW:.

Conclusioni

Questo articolo ha avuto lo scopo di illustrare la creazione di un'interfaccia tra i device Exec e l'AmigaDOS. L'impiego degli handler fornisce una maniera comoda per accedere ai device da un'applicazione scritta da un utente senza doversi complicare la vita con i reply port e le strutture. Inoltre, l'indipendenza da questi meccanismi laboriosi permette di ottenere un'interfacciamento omogeneo tra DOS e device Exec e semplifica drasticamente le operazioni di I/O.

Ringraziamenti

Vorrei ringraziare Matt Dillon, autore del programma DosDev (un esempio di RAM disk, reperibile sul Fish disk #113) a cui mi sono ispirato durante la stesura dello handler di esempio.

Anche il file dbg.c deriva dal programma di Matt, eccezion fatta per alcune modifiche necessarie per l'uso in questo progetto.

I programmi a corredo di questo articolo sono stati sviluppati con il l'assemblatore e compilatore C Manx Aztec 68K, versione 3.60a.

Listato 1: new-handler.h

```
#ifndef NEW-HANDLER_H
#define NEW-HANDLER_H  1

/****************************************************
 * new-handler.h
 *
 * header file generale per il nuovo handler
 *
 * Storia:
 *   12-09-88  creato

****************************************************/

#include <functions.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/devices.h>
#include <exec/exec.h>
#include <exec/io.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include <libraries/filehandler.h>
#include <ctype.h>
#include "mydev_def.h"


#define DOS_FALSE   0
#define DOS_TRUE    -1

/* comandi di azioni DOS action commands - esistono
   ma non sono definiti in dosextens.h */
#define ACTION_SEEK  1008L
#define ACTION_RAWMODE  994L

extern VOID *AbsExecBase;

extern VOID btos(), ReturnPkt();

extern SHORT handle_open(), handle_read(),
             handle_write(), handle_close();
extern BYTE *typetostr();


  /* definizioni di packet DOS addizionali */
#define ACTION_SET_RAW_MODE  ACTION_SCREEN_MODE
#define ACTION_FIND_INPUT  1005L
#define ACTION_FIND_OUTPUT  1006L
#define ACTION_END  1007L
#define ACTION_SEEK  1008L
#define MODE_READWRITE  1004L
#define MODE_READONLY  ACTION_FIND_INPUT
#define ACTION_OPENRW  MODE_READWRITE
#define ACTION_OPENOLD  ACTION_FIND_INPUT
#define ACTION_OPENNEW  ACTION_FIND_OUTPUT
#define ACTION_CLOSE  ACTION_END

  /* codici d'errore DOS addizionali */
#define ERROR_CANT_CREATE_REPLY     100L
#define ERROR_CANT_CREATE_REQUEST  101L
#define ERROR_CANT_OPEN_DEVICE      300L
#define ERROR_READ_ERROR            301L
#define ERROR_WRITE_ERROR           302L

  /* macro per gestire la conversione di puntatori
     da C a BCPL e viceversa */

#define BPTRtoCptr(Bp) ((BYTE *)((ULONG) (Bp) << 2))
#define CptrtoBPTR(Cp)  ((BPTR) ((ULONG) (Cp) >> 2))


#endif NEW-HANDLER_H
```

Listato 2: mydev_def.h

```
/****************************************************
 * mydev_def.h
 *
 * versione C del file mydev_def.i per mydev.device
 *
 * Storia:
 *    88-09-12  creato

****************************************************/

#ifndef MYDEV_DEF_H
#define MYDEV_DEF_H  1

#ifndef EXEC_TYPES_H
#include <exec/types.h>
#endif EXEC_TYPES_H
```

```
#ifndef EXEC_DEVICES_H
#include <exec/devices.h>
#endif EXEC_DEVICES_H

#ifndef EXEC_IO_H
#include <exec/io.h>
#endif EXEC_IO_H


#define MD_NUMUNITS   4

/* struttura per l'area dati del device mydev */
struct MyDev {
  struct Device     md_Device;
  struct ExecBase   *md_ExecBase;
  struct DosBase    *md_DosBase;
  APTR              md_SegList;
  UBYTE             md_Flags;
  UBYTE             md_pad;
  struct Unit       *md_Units[MD_NUMUNITS];
};

struct MyDevMsg {
  struct Message    mdm_Message;
  struct Device     *mdm_Device;
  struct Unit       *mdm_Unit;
  struct IOStdReq   *mdm_Iob;
};

struct MyDevUnit {
  UBYTE             mdu_UnitNum;
  UBYTE             mdu_pad;
  struct MyDevMsg   mdu_Msg;
  struct Process    *mdu_Process;
};

/* bit di stato e flag per l'unità fermata */
#define MDUB_STOPPED  2
#define MDUF_STOPPED  (1 << MDUB_STOPPED)

/* dimensione dello stack e priorità del processo
   che sarà creato */
#define MYPROCSTACKSIZE  0x400
#define MYPROCPRI        0

/* nome del mio device */
#define MYDEVNAME  "mydev.device"


#endif MYDEV_DEF_H
```

Listato 3: routs.c

```
/************************************************
 * routs.c
 *
 * Routine di comunicazione dell'handler;
 * comunicazioni con 'mydev.device'
 *
 * Storia:
 *  12-09-88  creato
 *
 * Compilazione:
 *          cc +c +d -b
 *
 ***********************************************/


#include "new-handler.h"


  /* variabili globali */
static struct IOStdReq    *req=NULL;    /* struttura
                             di richiesta al device */
static struct MsgPort     *reply=NULL;  /* Il device
                          mette le risposte in questa porta */
static BOOL dev_open=FALSE;
static UBYTE buff[100];

  /* variabili esterne */
extern struct DeviceNode *dn;       /* questo nodo
                             dell'handler del device */



/************************************************
 * handle_open()
 * apre il mydev.device per questo handler
 ***********************************************/


SHORT handle_open(pkt)

struct DosPacket *pkt;
{
  LONG error;
  UBYTE name[20], vol_name[20];
  USHORT i, len;

     /* nome da aprire */
  btos(pkt->dp_Arg3,buff);

     /* fa una copia e lo mette in maiuscolo */
  i = 0;
  while ((name[i] = toupper(buff[i])) != ':')  i++;
  name[i] = '\0';

     /* ottiene il nome del nodo del device; è già in
        maiuscolo */
  btos(dn->dn_Name,vol_name);
  len = strlen(vol_name);
  if (vol_name[len] == ':')
     len--;

  dbprintf("%s %s %s %d\n",buff,name,vol_name,len);

     /* stiamo cercando di aprire il file e non il
        device? */
  if (strncmp(name,vol_name,len) != 0)
     return(ERROR_DEVICE_NOT_MOUNTED);
```

```
/* il device è già aperto? */
  if (dev_open)
    return(ERROR_OBJECT_WRONG_TYPE);

    /* crea una reply port */
  reply = CreatePort("new_handler_reply",0);
  if (reply==NULL)
    return(ERROR_CANT_CREATE_REPLY);
    /* crea una struttura di richiesta */
  req = CreateStdIO(reply);
  if (req==NULL) {
    DeletePort(reply);
    reply = NULL;
    return(ERROR_CANT_CREATE_REQUEST);
  }

    /* apre il device */
  error = OpenDevice(MYDEVNAME,0L,req,0L);
  if (error) {
    DeleteStdIO(req);
    DeletePort(reply);
    req = reply = NULL;
    return(ERROR_CANT_OPEN_DEVICE);
  }

  dev_open = TRUE;

  return(0);

} /* fine di handle_open */

/**************************************************
 * handle_close()
 * chiude un device aperto - libera tutto ciò che vi
 * è associato
 **************************************************/

SHORT handle_close(pkt)
struct DosPacket *pkt;
{
  if (dev_open && req) {
    CloseDevice(req);
    dev_open = FALSE;
  }
  if (req) {
    DeleteStdIO(req);
    req = NULL;
  }
  if (reply) {
    DeletePort(reply);
    reply = NULL;
  }

  return(0);

} /* fine di handle_close */

/**************************************************
 * handle_read()
 * impartisce richieste di lettura a device di
 * sistema
 **************************************************/

SHORT handle_read(pkt)
struct DosPacket *pkt;
{
  if (req==NULL)
    return(ERROR_READ_ERROR);

  req->io_Command = CMD_READ;
  req->io_Data = (APTR)pkt->dp_Arg2;
  req->io_Length = pkt->dp_Arg3;

    /* impartisce la richiesta di io */
  DoIO(req);

  pkt->dp_Res1 = req->io_Actual;
  pkt->dp_Res2 = req->io_Error;

  return(0);

} /* fine di handle_read */

/**************************************************
 * handle_write()
 * impartisce una richiesta di scrittura a device di
 * sistema
 **************************************************/

SHORT handle_write(pkt)
struct DosPacket *pkt;
{
  if (req==NULL)
    return(ERROR_WRITE_ERROR);

  req->io_Command = CMD_WRITE;
  req->io_Data = (APTR)pkt->dp_Arg2;
  req->io_Length = pkt->dp_Arg3;

    /* impartisce la richiesta di io */
  DoIO(req);

  pkt->dp_Res1 = req->io_Actual;
  pkt->dp_Res2 = req->io_Error;

  return(0);

} /* fine di handle_write */

/* fine del file */
```

Listato 4: dbg.c

```
/**************************************************
 * dbg.c
 *
 * Codice di debugging per il DOS device handler
 *
 * Storia:
```

```
 *   21-08-88  creato da dosdevice.c di Matt Dillon
 *            (disponibile nel Pubblico Dominio)
 *
 * Compilazione:
 *               cc -n +c +d +b dbg
 ***************************************************/

#include "new-handler.h"


static struct MsgPort *Dback=NULL;     /* reply port
                                          di debug */
static struct MsgPort *Dbport=NULL;   /* reply port
                             di debug del processo */
static USHORT DBDisable=0;  /* flag per disabilitare
                                la stampa di debug */
static struct Message DummyMsg;    /* posseduto da
                                    sotto-processo */


extern struct DosLibrary *DOSBase;


#define DBG_PROC1_PORT_NAME  "dbg_reply_port_#1"
#define DBG_PROC2_PORT_NAME  "dbg_reply_port_#2"

/****************************************************
 * typetostr()
 * restituisce un messaggio del tipo di packet
 ***************************************************/

BYTE *typetostr(ty)
LONG ty;

{
    switch(ty) {
    case ACTION_CURRENT_VOLUME:
                    return("CURRENT VOL");
    case ACTION_DIE:             return("DIE");
    case ACTION_OPENRW:          return("OPEN-RW");
    case ACTION_OPENOLD:         return("OPEN-OLD");
    case ACTION_OPENNEW:         return("OPEN-NEW");
    case ACTION_READ:            return("READ");
    case ACTION_WRITE:           return("WRITE");
    case ACTION_CLOSE:           return("CLOSE");
    case ACTION_SEEK:            return("SEEK");
    case ACTION_EXAMINE_NEXT:
                    return("EXAMINE NEXT");
    case ACTION_EXAMINE_OBJECT:
                    return("EXAMINE OBJ");
    case ACTION_INFO:            return("INFO");
    case ACTION_DISK_INFO:
                    return("DISK INFO");
    case ACTION_PARENT:
                    return("PARENTDIR");
    case ACTION_DELETE_OBJECT:   return("DELETE");
    case ACTION_CREATE_DIR:
                    return("CREATEDIR");
    case ACTION_LOCATE_OBJECT:   return("LOCK");
    case ACTION_COPY_DIR:        return("DUPLOCK");
    case ACTION_FREE_LOCK:       return("FREELOCK");
    case ACTION_SET_PROTECT:
                    return("SETPROTECT");
    case ACTION_SET_COMMENT:
                    return("SETCOMMENT");
    case ACTION_RENAME_OBJECT:   return("RENAME");
    case ACTION_INHIBIT:         return("INHIBIT");
    case ACTION_RENAME_DISK:
                    return("RENAME DISK");
    case ACTION_MORE_CACHE:
                    return("MORE CACHE");
    case ACTION_WAIT_CHAR:
                    return("WAIT FOR CHAR");
    case ACTION_FLUSH:           return("FLUSH");
    case ACTION_RAWMODE:         return("RAWMODE");
    default:
                    return("----UNKNOWN----");
    }
}

/*
 *  CODICE DI DEBUGGING.   Non si può fare una
 * chiamata alla libreria DOS che acceda ad altri
 * device dall'interno di un device driver DOS
 * perché usa la stessa message port. del driver. Se
 * si ha bisogno di una chiamata di questo tipo
 * occorre creare la porta e costruirsi il messaggio
 * DOS da sé. Non ho fatto questo. Per ottenere
 * delle informazioni di debugging ho creato un
 * altro processo al quale possono essere mendati i
 * messaggi di debugging.
 *
 * Vorrete che la priorità del processo di debug sia
 * maggiore della priorità del vostro handler DOS.
 * Questo per fare in modo che, se il vostro handler
 * va in crash, possiate avere una migliore idea del
 * punto in cui è morto dai messaggi di debugging
 * (ricordate che i due processi sono asincroni
 * l'uno rispetto all'altro).
 */

extern VOID debugproc();

/****************************************************
 * dbinit()
 * inizializza il processo di debugging
 * eseguiamo il processo di debug a una priorità più
 * alta del task originante
 ***************************************************/

dbinit()
{
    struct Task *task = FindTask(NULL);

    Dback = CreatePort(DBG_PROC1_PORT_NAME,NULL);
    CreateProc("DEV_DB",
                  (LONG)task->tc_Node.ln_Pri+1,
                  (LONG)CptrtoBPTR(debugproc), 4096L);
    WaitPort(Dback);    /* partenza dell'handshake */
    GetMsg(Dback);  /* rimuove il messaggio dummy */
```

```
    dbprintf("Debugger running with %s\n\n",
             task->tc_Node.ln_Name);

} /* fine di dbinit */



/*****************************************************
 * dbuninit()
 * termina il processo di debugging
 *****************************************************/

dbuninit()
{
    struct Message killmsg;
    if (Dbport) {
      killmsg.mn_Length = 0;   /* 0 significa fine */
      PutMsg(Dbport,&killmsg);
      WaitPort(Dback);         /* è terminato */
      GetMsg(Dback);
      DeletePort(Dback);

   /*
    *  Poiché il processo di debugging è in
    *  esecuzione a una priorità più elevata, sono
    *  sicuro che è garantito che sia completamente
    *  rimosso prima che questo task riprenda
    *  nuovamente il controllo.
    */

      Delay(50L);  /* assicura che sia terminato */
    }

} /* fine di dbuninit */

/*****************************************************
 * dbprintf()
 * una routine tipo printf per stampare nella
 * finestra di debug
 *****************************************************/

dbprintf(a,b,c,d,e,f,g,h,i,j)
LONG a,b,c,d,e,f,g,i,j;
{
    BYTE buf[256];
    struct Message *msg;

    if (Dbport && !DBDisable) {
      sprintf(buf,a,b,c,d,e,f,g,h,i,j);
      msg = AllocMem((LONG)
              sizeof(struct Message)+strlen(buf)+1),
              MEMF_PUBLIC|MEMF_CLEAR);
      msg->mn_Length = strlen(buf)+1;
      strcpy(msg+1,buf);
      PutMsg(Dbport,msg);
    }

} /* fine di dbprintf */

/*****************************************************
 * debugmain()
 * la routine di debug principale - chiamata dal tag
 * assembler BTW, la DOS library usata da
 * debugmain() è stata aperta dal device driver.
 * Nota: DummyMsg non può essere sullo stack di
 * debugmain() perché debugmain() sparisce con
 * l'handshake finale.
 *****************************************************/

debugmain()
{
    struct Message *msg;
    SHORT len;
    VOID *fh;

    Dbport = CreatePort(DBG_PROC2_PORT_NAME,NULL);
    fh = Open("con:0/0/640/150/Debug Window",
         1006L);
    PutMsg(Dback, &DummyMsg);
    for (;;) {
      WaitPort(Dbport);
      msg = GetMsg(Dbport);
      len = msg->mn_Length;
      if (len == 0)
        break;
      --len;
      Write(fh, msg+1, (LONG)len);
      FreeMem(msg,
              (LONG)sizeof(struct Message)+len+1);
    }
    Close(fh);
    DeletePort(Dbport);
    PutMsg(Dback,&DummyMsg); /*termina l'handshake*/

} /* fine di debugmain */
/*****************************************************
 * debugproc()
 * Il tag assembly per il processo DOS: CNOP causa
 * dei problemi di allineamento con l'assembler
 * Aztec per qualche ragione. Ho assunto quindi che
 * l'allineamento sia sconosciuto. Poiché la
 * conversione BCPL di base azzera i due bit meno
 * significativi dell'indirizzo, il codice potrebbe
 * iniziare in un qualunque intorno
 * dell'etichetta...   Sigh...
 *****************************************************/

#asm
   public   _debugproc
   public   _debugmain

   cseg
   nop
   nop
   nop
_debugproc:
   nop
   nop
   movem.l D2-D7/A2-A6,-(sp)
```

```
   jsr    _debugmain
   movem.l (sp)+,D2-D7/A2-A6
   rts
#endasm

/* fine del file */
```

Listato 5: new-handler.c

```
/*****************************************************
 * new-handler.c
 *
 * Codice del device handle AmigaDOS d'esempio
 *
 * (c) Copyright 1989 Steve Simpson
 *                    Skarpskyttevagen 20C
 *                    S-222 42 Lund, SWEDEN
 *
 * Storia:
 *  12-09-88  creato
 *  13-09-88  nessun volume node
 *
 * compilazione:
 *             cc +c +d +b -n new-handler
 * link:
 *             ln -g -w new-handler routs dbg -lcl
 ****************************************************/

#include "new-handler.h"

  /* potete ridefinire queste linee */
#define DEBUG  1
#define VOLUME  0

  /* variabili esterne */
extern struct DosLibrary      *DOSBase;
extern struct ExecBase        *SysBase;

  /* variabili globali */
struct Process        *DevProc;    /* questo device
                                      handler */
struct MsgPort        *DevPort;
struct Message        *msg;
struct DosInfo        *di;
struct DeviceList     *dl;
struct DeviceNode     *dn;
struct DosPacket      *pkt;        /* packet ricevuto
                                      dal DOS */

/*****************************************************
 * handler()
 * Questa è la routine principale del modulo -
 * inizializza i puntatori alla libreria, risponde
 * al packet iniziale del DOS e direge le richieste
 * alle routine appropriate. Non abbiamo chiamato
 * questa routine main() in modo da sapere se è
 * stato fatto qualche errore quando arriva il
 * momento del link.
 ****************************************************/

VOID handler()
{
  UBYTE str[100];

    /* inizializza le variabili globali */
  SysBase = (struct ExecBase *)AbsExecBase;
  DevProc = (struct Process *)FindTask(0L);
                                   /* questo task */
  DOSBase = (struct DosLibrary *)
               OpenLibrary("dos.library",0L);

    /* aspetta il packet iniziale dal DOS */
  DevPort = &DevProc->pr_MsgPort;
  WaitPort(DevPort);
  msg = GetMsg(DevPort);
  pkt = (struct DosPacket *)msg->mn_Node.ln_Name;

    /* dovremmo impostare DosNode->dn_Task a un
     * valore non-NULL, altrimenti viene creata una
     * nuova istanza del device per ogni
     * riferimento.
     */
  if (DOSBase) {
    di = (struct DosInfo *)
          BPTRtoCptr(((struct RootNode *)
          DOSBase->dl_Root)->rn_Info);

#if VOLUME==1
      /* crea un volume node */
    dl = (struct DeviceList *) AllocMem((LONG)
          sizeof(struct DeviceList),
          MEMF_PUBLIC | MEMF_CLEAR);
#endif VOLUME

      /* ptr.al device node */
    dn = (struct DeviceNode *)
          BPTRtoCptr(pkt->dp_Arg3);

#if VOLUME==1
      /* 'mount' creerà un device node per noi; se
       * stiamo creando un file system device
       * potremmo avere bisogno di includere un
       * volume node così che il WB/DOS ci
       * riconosca come un device. Questo pezzetto
       * di codice è incluso come un esempio e non
       * ha nessuna rilevanza qui, visto che ci
       * stiamo occupando solo di creare un device
       * (eseguite 'assign' dal CLI per capire
       * cosa voglio dire).
       */
    dl->dl_Type     = DLT_VOLUME;
    dl->dl_Task     = DevPort;
    dl->dl_DiskType = ID_DOS_DISK;
    dl->dl_Name     = dn->dn_Name;
    dl->dl_Next     = di->di_DevInfo;
    di->di_DevInfo  = (LONG)CptrtoBPTR(dl);
#endif VOLUME

      /* imposta il campo dn_Task - questo indica
       * al DOS di non creare un nuovo processo
```

```
         * per ogni riferimento
         */
      dn->dn_Task = DevPort;
      pkt->dp_Res1 = DOS_TRUE;
      pkt->dp_Res2 = 0;
      ReplyPkt(pkt);
   }

   else {
      pkt->dp_Res1 = DOS_FALSE;
      ReplyPkt(pkt);
      return;
   }

#if DEBUG==1
    /* inizializza il codice di debugging */
  dbinit();
#endif DEBUG

    /* qui inizia il loop senza fine - aspetta le
     * richieste dalla message port e le esegue
     */

  do_control();
    /* bisogna arrivare qui per rimuovere l'handler;
     * si avrà bisogno di impostare qualche
     * condizione di uscita all'interno del driver
     * per uscire dal loop di controllo e arrivare
     * qui.
     */
  remove_handler();

#if DEBUG==1
    /* rimuove il sotto-processo di debug */
  dbuninit();
#endif DEBUG

  cleanup();

} /* fine di handler */



/****************************************************
 * do_control()
 * la routine loop (senza fine!).  Riceve i
 *  susseguenti packet dal DOS e li gestisce.
 ****************************************************/

do_control()
{
  USHORT quit=FALSE;
  SHORT error;
  struct Task *new_task;
top:
  quit = FALSE;

  while (!quit) {

      WaitPort(DevPort);
      while (msg = GetMsg(DevPort)) {
         pkt = (struct DosPacket *)
               msg->mn_Node.ln_Name;

         pkt->dp_Res1 = DOS_TRUE;
         pkt->dp_Res2 = 0;

#if DEBUG==1
         dbprintf(
           "Packet: %4ld %06lx %06lx %06lx %10s\n",
            pkt->dp_Type,pkt->dp_Arg1,pkt->dp_Arg2,
            pkt->dp_Arg3, typetostr(pkt->dp_Type));
#endif DEBUG

           /* fa uno switch sul tipo del packet
            * gli argomenti sono messi nella forma:
            *  Arg:arg1,arg2,arg3,arg4
            *  Ris:ris1,ris2
            */
         switch(pkt->dp_Type) {

         case ACTION_DIE:
             quit = TRUE;
             break;

             /* Arg: FileHandle,Lock,Nome
                Ris: Booleano */
         case ACTION_OPENRW:
         case ACTION_OPENOLD:
         case ACTION_OPENNEW:
             error = handle_open(pkt);
             break;

             /* Arg: FileHandle,Buffer,Lunghezza
                Ris: Lunghezza effettiva */
         case ACTION_READ:
             error = handle_read(pkt);
             break;

             /* Arg: FileHandle,Buffer,Lunghezza
                Ris: Lunghezza effettiva */
         case ACTION_WRITE:
             error = handle_write(pkt);
             break;

             /* Arg: FileHandle
                Ris: Booleano */
         case ACTION_CLOSE:
             error = handle_close(pkt);
             break;

             /* Arg: Lock,Nome,Modo
                Ris: Lock */
         case ACTION_LOCATE_OBJECT:
             error = ERROR_BAD_STREAM_NAME;
             break;

             /* Arg: FileHandle,Posizione,Modo
                Ris: Vecchia posizione */
         case ACTION_SEEK:
```

```
            /* Arg: Lock,FileInfoBlock
               Ris: Booleano */
        case ACTION_EXAMINE_NEXT:

            /* Arg: Lock,FileInfoBlock
               Ris: Booleano */
        case ACTION_EXAMINE_OBJECT:

            /* Arg: Lock,Nome
               Ris: Boolean */
        case ACTION_DELETE_OBJECT:

            /* Arg: Lock,Nome
               Ris: Boolean */
        case ACTION_CREATE_DIR:

        case ACTION_COPY_DIR:

            /* Arg: Lock,InfoData
               Ris: Booleano */
        case ACTION_INFO:

            /* Arg: InfoData
               Ris: Booleano */
        case ACTION_DISK_INFO:

            /* Arg: LockDa,NomeDa,LockA,NomeA
               Ris: Booleano */
        case ACTION_RENAME_OBJECT:

            /* Arg: Lock
               Ris: Boolean */
        case ACTION_FREE_LOCK:

        case ACTION_FLUSH:

        case ACTION_RENAME_DISK:

        default:
            pkt->dp_Res1 = DOS_FALSE;
            pkt->dp_Res2 = ERROR_ACTION_NOT_KNOWN;
            break;

        } /* switch */

        if (error) {
          pkt->dp_Res1 = DOS_FALSE;
          pkt->dp_Res2 = error;
        }

#if DEBUG==1
        dbprintf(
     "ARG: %06lx %06lx %06lx  RIS: %06lx %06ld\n",
           pkt->dp_Arg1,pkt->dp_Arg2,pkt->dp_Arg3,
           pkt->dp_Res1,pkt->dp_Res2);
#endif DEBUG

        ReplyPkt(pkt);

    } /* while */

  } /* while */

#if DEBUG==1
  new_task = (struct Task *)DevProc;
  dbprintf("Can we remove %s?  ",
           new_task->tc_Node.ln_Name);
#endif DEBUG
  Delay(50L);
  Forbid();
  if (PktsQueued(DevProc)) {
      Permit();

#if DEBUG
      dbprintf("... not yet!\n");
#endif DEBUG

      goto top;
  }
  Permit();
  return(0);

} /* fine di do_control */


/****************************************************
 * remove_handler()
 * rimuove questo handler dalle lista dei nodi
 * device/handler mantenuta dall'AmigaDOS
 ***************************************************/
remove_handler()
{
  VOID *dlp;
  struct DosInfo *dsi;
  struct DeviceList *dvl;

  dn->dn_Task = NULL;

    /* rimuove questo volume node */
  dsi = (struct DosInfo *)
         BPTRtoCptr(((struct RootNode *)
         DOSBase->dl_Root)->rn_Info);
  dlp = &dsi->di_DevInfo;
  for (dvl = (struct DeviceList *)
         BPTRtoCptr(dsi->di_DevInfo);
       dvl && dvl != dl;
       dvl = (struct DeviceList *)
             BPTRtoCptr(dvl->dl_Next)) {
    dlp = &dvl->dl_Next;
  }
  if (dvl == dl) {
    *(BPTR *)dlp = dvl->dl_Next;
    FreeMem(dvl,(LONG)sizeof(struct DeviceList));
  }
#if DEBUG==1
  else {
    dbprintf(" YAH!!! - Where's the volume node\n");
  }
#endif DEBUG

} /* fine di remove_handler */
```

```
/*****************************************************
* cleanup()
* chiude tutto quello che è stato aperto
*****************************************************/

cleanup()
{
  if (DOSBase != NULL)
    CloseLibrary(DOSBase);

} /* fine di cleanup */


/*****************************************************
 * btos()
 * Converte un BSTR in una normale stringa C
 *****************************************************/

VOID btos(bstr,buf)
UBYTE *bstr;
UBYTE *buf;
{
    bstr = (UBYTE *)BPTRtoCptr(bstr);
    BCPL_CString(bstr,buf);

} /* fine di btos */


/*****************************************************
 * PktsQueued()
 * Ci sono dei packet in coda al nostro device?
 *****************************************************/

PktsQueued(dp)
struct Process *dp;
{
  return ((VOID *)dp->pr_MsgPort.mp_MsgList.lh_Head
!=
    (VOID *)&dp->pr_MsgPort.mp_MsgList.lh_Tail);

} /* fine di PktsQueued */


/*****************************************************
 * ReplyPkt()
 * risponde a un packet DOS; spedisce indietro i
 * packet al DOS
 *****************************************************/

ReplyPkt(pkt)
register struct DosPacket *pkt;
{
  register struct Message *mess;
  register struct MsgPort *replyport;

    /* copia la message reply port */
  replyport = pkt->dp_Port;
  mess = pkt->dp_Link;
  pkt->dp_Port = DevPort;
  mess->mn_Node.ln_Name = (BYTE *)pkt;
  mess->mn_Node.ln_Succ = NULL;
  mess->mn_Node.ln_Pred = NULL;

  PutMsg(replyport,mess);

} /* fine di ReplyPkt */


/*****************************************************
 * BCPL_CString()
 * Assembla una stringa C da una stringa BCPL
 * Una stringa BCPL assomiglia a questo:
 *
 *   lung. char  char  char
 *  !-----!-----!-----!-----!
 *   byte0 byte1 byte2 byte3  ecc.
 *****************************************************/


BCPL_CString(bcpl,cstr)
UBYTE *bcpl,  /* ptr alla stringa BCPL */
      *cstr;  /* per la stringa restituita */
{
  USHORT len;

  len = *(bcpl+0);    /* il primo byte fornisce la
                         lunghezza della stringa */
  if (len==0) {
    *(cstr+0) = '\0';
    return(0);
  }
  strncpy(cstr,(bcpl+1),len);
  *(cstr+len) = 0;
  return(1);

} /* fine di BCPL_CString */


/* fine del file */
```

Listato 6: test_new.c

```
/*****************************************************
 * test_new.c
 *
 * Testa le chiamate del nuovo handler (NEW: device)
 *
 * Storia:
 *    88-11-09  creato
 *****************************************************/

#include <functions.h>
#include <exec/types.h>
#include "new-handler.h"

#define NEW_NAME  "NEW:"

struct FileHandle *fh;
UBYTE buff[20];
```

```
main()
{
  LONG len;

      /* apre il device NEW: - la mountlist fornisce
       * l'indicazione su dove si trovi il device
       * handler
       */
  fh = Open(NEW_NAME, MODE_NEWFILE);
  if (fh==NULL) {
    printf("Can't open DOS device %s\n",NEW_NAME);
    exit(0);
  }
  else
    printf("DOS device %s opened successfully\n",
           NEW_NAME);

      /* cerca di leggere dei dati dal device driver
       */
  len = Read(fh, buff, (LONG)sizeof(buff));
  printf("Read: Actual=%ld\n",len);

      /* ora scrive dei dati al device driver */
  len = Write(fh, buff, (LONG)sizeof(buff));
  printf("Write: Actual=%ld\n",len);

      /* chiude il DOS device */
  Close(fh);

} /* fine di main */

/* fine del file */
```

Listato 7: do_mount

```
.key directory
if "<directory> eq ""
echo "Uso: execute do_mount [assign directory]"
skip exit
endif

assign devs: <directory>
assign l: <directory>
mount "new:"
binddrivers
echo "Device NEW: mounted"

lab exit
```

Listato 8: mountlist

```
/* mountlist per il nuovo device handler AmigaDOS */
NEW:       Handler = L:new-handler
           Stacksize = 3000
           Priority = 5
           GlobVec = 1
#
```

(segue da pagina 36)

Obbedire alle regole!

Si riscontra un atteggiamento insistente in molti gruppi di programmazione, i quali sembrano accettare il fatto che, una volta che il software non va in crash troppo spesso e che tutte le opzioni sono velocemente accessibili, l'interfaccia utente è stata rispettata. È la stessa mentalità che parteggia per la produzione di cianfrusaglie piuttosto che per un prodotto serio e basilare.

È ancora tutto così lontano dalla realtà e ci vorrebbe uno studio approfondito sotto il punto di vista psicologico, ma questo avverrà in seguito.

La filosofia della Commodore

La Commodore spiegò la sua filosofia nella prima edizione del manuale Intuition:

"Che cosa è l'interfaccia utente? Questa frase così generica comprende tutti gli aspetti che riguardano gli input e gli output tra l'utente e la macchina. Include i meccanismi più interni del computer e arriva fino al punto di definire una filosofia per guidare il rapporto tra l'uomo e la macchina. Intuition è, soprattutto, una filosofia trasformata in software."

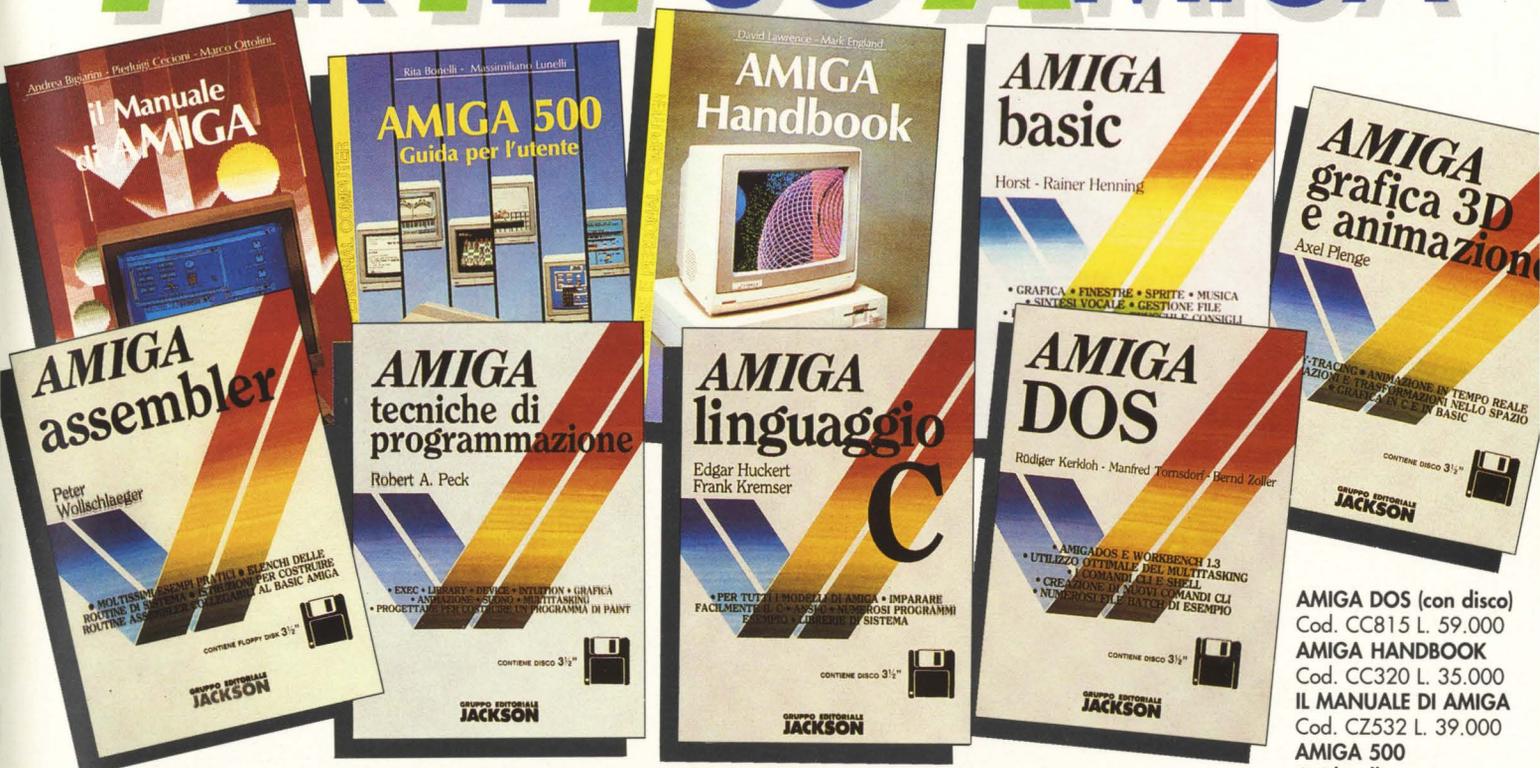
"È facile descrivere questa filosofia: l'interazione tra il computer e l'utente dovrebbe essere semplice, divertente e coerente, detto in una parola, intuitiva. Intuition fornisce un insieme di strumenti e di ambienti che possono essere utilizzati per andare incontro a questa filosofia."

"Vi si incoraggia a sfruttare le varie caratteristiche di Intuition. Facendo questo raggiungerete due scopi: passerete meno tempo ad implementare per conto vostro i meccanismi di interazione utente... e l'utente del vostro prodotto si troverà a lavorare in un ambiente che non cambia radicalmente da un'applicazione all'altra."

"Non importa quanto il vostro programma sia semplice o fantasioso, si adatterà alle regole base dell'ambiente Intuition... L'utente apprenderà gli elementi base di Intuition ed imparerà ad avere fiducia nel fatto che i blocchi centrali resteranno tali. Questa coerenza assicura che un programma ben progettato verrà compreso dall'utente principiante come pure da quello sofisticato."

"Questa è l'essenza e la bellezza della filosofia di Intuition"

Se solo le software house rispettassero questo e mettessero da parte la loro presunzione e la loro arroganza... Il software è per l'utente, non per l'ego della società autrice!

GRANDE CONCORSO 9 LIBRI JACKSON PER IL TUO AMIGA

AMIGA DOS (con disco)
Cod. CC815 L. 59.000
AMIGA HANDBOOK
Cod. CC320 L. 35.000
IL MANUALE DI AMIGA
Cod. CZ532 L. 39.000
AMIGA 500
Guida all'acquisto
Cod. CC627 L. 55.000
AMIGA ASSEMBLER (con disco)
Cod. CL757 L. 59.000
AMIGA LINGUAGGIO C (con disco)
Cod. CL758 L. 52.000
AMIGA GRAFICA 3-D (con disco)
Cod. CZ756 L. 59.000
AMIGA BASIC (con disco)
Cod. CL768 L. 57.000

AMIGA - Tecniche di programmazione (con disco)
Cod. CC795 L. 62.000

PER TE FAVOLOSI PREMI

Ecco una nuova straordinaria iniziativa del Gruppo Editoriale Jackson: è lo speciale concorso riservato a te e dedicato al tuo fantastico Amiga.

Partecipare è facile: basta acquistare uno dei nove libri Jackson per i computer Amiga, compilare la speciale cartolina che trovi dal tuo rivenditore di fiducia e spedirla.

E' facile anche vincere e i premi in palio sono fantastici: un personal computer Commodore Amiga 2000, una stampante a colori Commodore MPS 1500C e 8 set di programmi software della serie "Software Commodore by CTO".

GRUPPO EDITORIALE JACKSON

AUT. MIN. RICH.

BORN IN THE USA
COMPUTE!'s AMIGA RESOURCE
SPECIAL PRICE $2.95
SPECIALE DESKTOP VIDEO
MAGAZINE
AMIGA
IL MENSILE JACKSON PER GLI UTENTI DI AMIGA